Aplicación para memorizar usando repetición espaciada

Ana María Hernández Gómez

Enero 2013

Índice general

Jose Antonio Rodríguez Garrido
Director del proyecto

Toni Cortés Roselló
Tutor del proyecto

Ana Mª Hernández
Autora del proyecto

Resumen

El aprendizaje es un proceso a través del cual se adquieren o se modifican habilidades y conocimientos como resultado del estudio, la observación y el razonamiento[1]. Una parte del aprendizaje se centra en la memorización de conceptos que serán utilizados en un futuro más o menos cercano. Cuando no se hace un uso de estos conceptos a corto o a medio plazo, es fácil observar que cuesta recordarlos y, en realidad, es como si no se hubieran aprendido: se han olvidado fácilmente. Así pues, se tiende a afirmar que gracias a la práctica se puede adquirir un concepto de forma duradera.

Por este motivo, la finalidad de este proyecto es el desarrollo de una herramienta que permita el aprendizaje y memorización de contenido diverso centrándose en la repetición de ejercicios de forma espaciada en el tiempo[2], haciendo del aprendizaje una tarea más fructífera y agradable al usuario. La repetición de los ejercicios de forma espaciada es lo que hará que dichos conceptos se mantengan por más tiempo en la memoria. Dicha repetición tiene en cuenta los aciertos y fallos del usuario que está resolviendo un problema para así determinar en qué momento ha de volver a practicar ese problema. En definitiva, la práctica continuada de un concepto nuevo hasta que se convierte en un concepto conocido y recordado por el usuario es lo que realmente facilita el aprendizaje.

El sistema es flexible a la hora de ampliar el contenido disponible para el estudio, pues se ha basado su diseño en la orientación a objetos. Se ha centrado en el aprendizaje de vocabulario en inglés permitiendo aprender una palabra, su traducción, su pronunciación, su escucha y su relación con un símbolo o imagen.

Siguiendo el manual del desarrollador, que se puede consultar en el anexo A, se puede realizar una continuación del proyecto y ampliar de las funcionalidades del mismo.

[1] Más información en: `http://es.wikipedia.org/wiki/Aprendizaje` .
[2] Más información en: `http://es.wikipedia.org/wiki/Repaso_espaciado` .

Dedicado a mis padres, a mi hermano y a toda mi familia por ser la mejor familia del mundo entero.

Especialmente a Jordi por toda su paciencia, sus ganas de ayudar, su apoyo y su infinita comprensión durante el desarrollo del proyecto...

Y finalmente a mis compañeros y amigos de facultad, especialmente a Marc, por ayudarme a corregir y a mejorar este proyecto y a Alex, por aportarme claridad de ideas en el extenso mundo de las licencias de *software*.

Gracias.

Capítulo 1

Introducción

El aprendizaje es la adquisición mediante la práctica de una conducta duradera[1]. Sin embargo, la memorización no es un proceso sencillo de llevar a cabo y muchas veces no es agradable. Cuando es necesario aprender una lección para un examen, se intentan memorizar aquellas partes más importantes de la misma resumiéndola, haciendo esquemas, realizando algún dibujo o asociando algún mnemotécnico[2] para facilitar el recuerdo más adelante.

Sin embargo, suele ocurrir que cuando estos conceptos recién adquiridos son utilizados para responder correctamente dicho examen y posteriormente no vuelven a ser requeridos (es decir, que no se hace uso de la memoria), resulta que finalmente no se pueden recordar y en realidad es como si no se hubieran aprendido.

El resultado de este proyecto es un sistema *software* que se centra en facilitar al usuario las herramientas necesarias para que el aprendizaje sea sencillo y perdure en el tiempo a base de presentarle unos ejercicios que serán planificados en el tiempo según las respuestas obtenidas. El sistema recibe el nombre de **ULearn**.

El desarrollo de este proyecto se ha realizado siguiendo la metodología *Rational Unified Process* o `RUP` [15], la cual es una metodología iterativa de desarrollo. En ella se plantea la división del proyecto en fases, las cuales pueden tener una o más iteraciones. Las fases de desarrollo son cuatro: fase inicial, de elaboración, de construcción y de transición. La documentación generada en cada una de estas fases se ha dispuesto adecuadamente en los diferentes capítulos que forman la memoria de este proyecto [14].

[1] Definición: `http://rae.es/aprendizaje` .
[2] Definición: `http://rae.es/mnemotecnia` .

Capítulo 2

Visión general del proyecto

2.1. Descripción del problema

El estudio y la adquisición de conocimiento es una actividad que, de una forma u otra, nos acompaña a lo largo de toda la vida. Si bien es cierto que hay diferentes maneras de adquirir dicho conocimiento, ya sea a base de teoría o de práctica, éste sólo quedará retenido en la memoria cuando se haya estudiado o practicado reiteradamente en un período de tiempo determinado.

Existe, por lo tanto, un momento en el tiempo en el que si no se vuelven a estudiar o a practicar estos conocimientos se acaban olvidando o, sencillamente, se hacen más difíciles de recordar. Así pues, la repetición de los ejercicios de forma espaciada es lo que hace que dichos conceptos se mantengan por más tiempo en la memoria.

Este no es un concepto nuevo puesto que allá por 1885 Hermann Ebbinghaus[1] publicó un libro llamado *Über das Gedächtnis* (traducido posteriormente al inglés como *Memory. A Contribution to Experimental Psychology*), en el cual detalló los experimentos que probó en sí mismo para describir los procesos de aprendizaje y de olvido. Se le considera un pionero en el estudio de la memoria y es conocido por ser la primera persona que describió el significado de la curva de aprendizaje y la del olvido, así como el efecto del espaciado en el tiempo.

Siguiendo la línea que trazó en un primer momento Hermann Ebbinghaus, en 1970 Sebastian Leitner[2], un divulgador y científico alemán que se centró en el estudio de temas médicos y psicológicos, diseñó un método de aprendizaje[3] basado en el uso de tarjetas de preguntas y respuestas que permitían acortar el tiempo de estudio a medio y largo plazo. En la figura 2.1 se muestra un esquema del método de Leitner en el cual se observa que en el momento de empezar el estudio todas las tarjetas se colocan en la caja número 1 y, a medida que se van acertando, se colocan en la caja número 2, número 3 y así sucesivamente. En caso de fallar una tarjeta se coloca siempre en la primera caja. El estudio se realiza espaciadamente de forma que el primer día se estudian las tarjetas pertenecientes a la caja número 1; al día siguiente se vuelven a estudiar las tarjetas de la caja número 1 y dos días después se repasan las de la caja número 2. Entre tres y cinco días después de haber iniciado el estudio, se repasan las de la caja número 3 y así sucesivamente.

[1]Más información en: `http://en.wikipedia.org/wiki/Hermann_Ebbinghaus` .
[2]Más información en: `http://es.wikipedia.org/wiki/Sebastian_Leitner` .
[3]Más información en: `http://en.wikipedia.org/wiki/Leitner_System` .

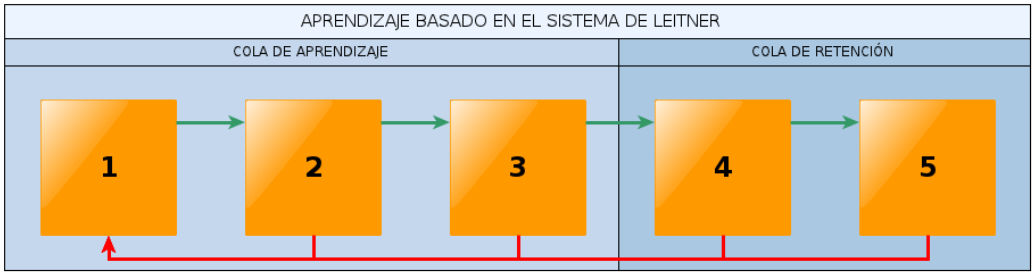


Figura 2.1: Esquema del Método de Leitner.

Una variante de este método es aquella que permite que las tarjetas falladas no se reinicien a una posición inicial, si no a la inmediatamente anterior, permitiendo así no penalizar de forma muy acusada tarjetas que ya se han estudiado con anterioridad (figura 2.2).

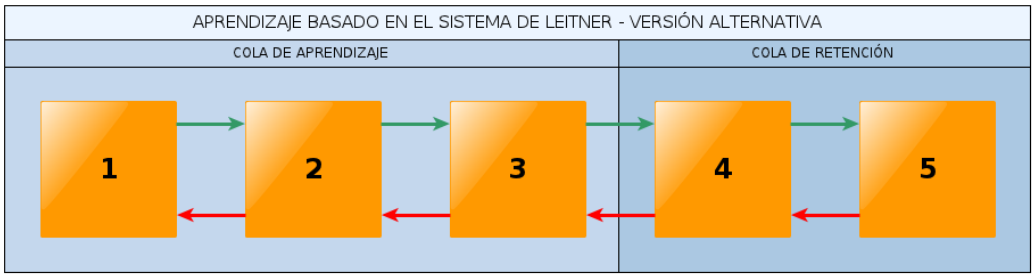


Figura 2.2: Esquema alternativo del Método de Leitner.

A partir de la idea de Leitner, en 1985 Piotr A. Woźniak[4] desarrolla un algoritmo basado en la repetición espaciada de ejercicios llamado *SuperMemo* [17]. Seguidamente aparecen más programas orientados en este sentido para así hacer del estudio una tarea más sencilla.

La única problemática es que gran parte de estos programas se centran en la opinión del usuario con respecto al ejercicio que acaba de resolver, en cómo de difícil le ha resultado. El impacto asociado en este sentido es que el aprendizaje se hace más lento, pues el usuario ha de pensar y valorar qué nivel de dificultad ha supuesto para él cada ejercicio.

Por este motivo, la solución propuesta es la de desarrollar una herramienta útil a tal efecto, haciendo un cálculo automático para realizar la repetición de un ejercicio estudiado según la respuesta dada por el usuario, sin que éste tenga que hacer ninguna valoración del ejercicio que acaba de resolver. Esta idea no es completamente novedosa puesto que ya existen otras herramientas que lo implementan de una manera similar, tal y como se explica en el apartado 2.6.4.

A través de este método, una resolución rápida y correcta de un ejercicio hará que éste sea repetido a más largo plazo que una resolución lenta o incorrecta. Así pues, se puede afirmar que los ejercicios se planifican según el progreso del usuario en el aprendizaje de un tema. Además, la herramienta es fácilmente ampliable en cuanto a la variedad de material disponible para el estudio.

[4]Más información en: `http://www.supermemo.com/english/company/wozniak.htm` .

Se considerará que el usuario ha aprendido un concepto o conoce un problema por completo cuando conozca todos los aspectos del mismo. El desarrollo de este proyecto se basa principalmente en el aprendizaje de vocabulario en inglés. Así pues, se han definido cinco aspectos para cada problema (en este caso son palabras) que el usuario debe aprender y memorizar: traducción de la palabra, traducción en sentido inverso, pronunciación, relación con un símbolo y finalmente la escucha de un audio de dicha palabra que el usuario debe escribir correctamente.

El proyecto presentado puede servir como base para futuras aportaciones o mejoras como se detallará más adelante.

2.2. Finalidad del proyecto

La finalidad de este proyecto es el desarrollo de un sistema *software* que se centre en facilitar al usuario las herramientas necesarias para aprender nuevos conceptos de manera sencilla y perdurable en el tiempo, a base de la realización de ejercicios que se repiten en un intervalo de tiempo determinado según las respuestas obtenidas. El sistema está diseñado para que sea fácilmente ampliable y está documentado debidamente a tal efecto en el apartado A.4.

2.3. Objetivos generales

Los objetivos a cumplir en este proyecto son:

- Ofrecer un servicio mediante una aplicación web para el aprendizaje espaciado en el tiempo.
- Garantizar que la aplicación tiene unas mínimas condiciones de diseño y usabilidad para hacerla agradable al usuario.
- Planificar adecuadamente los ejercicios respecto de cada usuario y sus respectivas respuestas.
- Controlar que la planificación se ha producido en el intervalo estimado para garantizar así el buen funcionamiento de la aplicación.
- Permitir que la aplicación sea fácilmente modificable o ampliable, basando el diseño en este supuesto.
- Documentar de forma clara el código de la aplicación, permitiendo así que futuros desarrolladores puedan continuar este proyecto de una forma sencilla.

En resumen, se pretende crear una aplicación que permita al usuario estudiar de forma espaciada en el tiempo que cumpla una sencilla regla: que sea útil, utilizable y utilizada. Esto es:

- **útil**: desde un enfoque analítico, se puede determinar que la aplicación aporta un valor al usuario.
- **utilizable**: desde un enfoque de diseño, se determina que la aplicación será lo suficientemente sólida como para ser utilizada.

- **utilizada**: el paso final, del prototipo a la puesta en marcha. Conseguir que la aplicación sea utilizada por cualquier usuario. En este momento no se puede hacer una valoración de este aspecto, puesto que la herramienta se ha diseñado inicialmente como proyecto final de carrera.

2.4. Posicionamiento del producto

El sistema está pensado para todas aquellas personas que quieran aprender por sí mismas una materia determinada, o que quieran reforzar sus conocimientos con ejercicios diversos realizando un análisis y seguimiento del progreso de cada usuario. La tabla 2.1 resume el posicionamiento del producto en el mercado.

Destinatario	**Concreción**
Para...	todo aquél que desee aprender algo y/o que desee participar en la creación de cursos.
Quienes...	desean mejorar la experiencia en el proceso de aprendizaje.
El proyecto es...	un programa; es decir, un producto *software*.
Y, además, ...	es suficientemente flexible para poder añadir funcionalidades que extiendan el material disponible al estudio, que permitan mejorar la experiencia del usuario o incluso que permitan cambios estéticos y visuales.
No es como...	las técnicas de aprendizaje habituales, las cuales no son eficientes a largo plazo (se olvidan los conceptos más rápidamente al no volver a repasar aquello que se ha dado por superado).
Ya que este proyecto...	se basa en análisis del conocimiento de un usuario respecto la resolución de problemas y, como tal, se trata de un proceso complejo y no exento de errores y/o imprecisiones.

Cuadro 2.1: Posicionamiento del producto.

2.5. Descripción de los usuarios

Identificar adecuadamente a los usuarios de la herramienta no es una tarea sencilla ni tampoco es algo trivial. Detallar las necesidades que aparecerán conociendo desde un principio qué tipo de usuarios pueden requerir este servicio proporciona información clave a la hora de definir en sí misma la herramienta y los requisitos que se han de tener en cuenta en su elaboración. Es por esto que el análisis de los posibles usuarios del sistema ha de formar parte del modelado.

2.5.1. Resumen de las partes interesadas en el desarrollo

La tabla 2.2 muestra las partes interesadas en el desarrollo del proyecto informático.

Nombre	Representa a...	Rol
Usuario	Persona interesada en el uso de una herramienta que facilite el autoaprendizaje de una determinada materia y en la ampliación de dichas materias de estudio.	Interactuar con la aplicación. Puede pedir nuevo material a ser incluido en los programas de estudio.
Técnico	Persona encargada del desarrollo del sistema. Posteriormente, de mantenerlo y actualizarlo.	Desarrollar el sistema y comprobar que los requisitos del mismo se cumplen según se han definido.
Gestor	Persona responsable de los servicios que ofrece el sistema y de la toma de decisiones.	Decidir qué materiales se incluyen y cuáles no como sesiones de estudio. Aprobar las distintas funcionalidades que tendrá el sistema y supervisar actualizaciones del mismo.

Cuadro 2.2: Personas interesadas en el desarrollo del sistema.

2.5.2. Resumen de los usuarios

La tabla 2.3 muestra una descripción de los usuarios que utilizarán la aplicación.

Nombre	Responsabilidades	Representado por...
Usuario	Disponer de una herramienta que permita un uso fluido y asegure un funcionamiento correcto.	Un usuario cualquiera.
Técnico	Solucionar posibles problemas que puedan reportar los usuarios y analizar las necesidades de los mismos según las sugerencias que aporten. Mantener el sistema.	El técnico que finalmente ha desarrollado y administrado este proyecto.
Gestor	Persona que posee todas las responsabilidades del Técnico además de la gestión de usuarios y de la creación de cursos.	Un usuario con privilegios totales sobre la aplicación.

Cuadro 2.3: Personas que interactúan con la aplicación.

2.5.3. Entorno y restricciones del sistema

El entorno es virtual; es decir, el usuario accede a la herramienta de forma *online* con un navegador cualquiera. Las restricciones que definen el sistema son las siguientes:

- Existe un único técnico que desarrollará el sistema planteado. Los perfiles que necesitará aplicar, dependiendo de la etapa en la que se encuentre, serán: jefe de proyecto, analista y arquitecto.

- Se requerirá autenticación por parte del usuario para poder acceder a la aplicación.

- La introducción de materiales al sistema la realizará el técnico o el gestor.

- Se espera que este proyecto permita añadir nuevos cursos de aprendizaje, para así ampliar el catálogo de materias disponibles para el usuario. A tal efecto, los usuarios serán libres de aportar materiales que crean que son interesantes para el estudio. Posteriormente, se moderarán y publicarán aquellos que el técnico, junto con el gestor, decidan que son adecuados.

- Este proyecto se realizará a través de un proyecto de final de carrera de Ingeniería en Informática, bajo la supervisión de la Facultad de Informática de Barcelona[5], por lo que se rige por la normativa vigente actual para los proyectos de final de carrera[6].

2.5.4. Necesidades de los usuarios

En la tabla 2.4 se muestran los problemas que existen actualmente para los usuarios y si existe alguna solución al respecto.

2.6. Visión general del producto

Este sección describe a alto nivel las características del sistema **ULearn**, interfaces y otras aplicaciones, así como configuraciones del propio sistema.

2.6.1. Perspectiva del producto

Este proyecto es una creación desde cero con el objetivo de obtener una herramienta que permita poder realizar ejercicios que refuercen el aprendizaje de una determinada materia, con lo que es totalmente independiente a cualquier otro. El planteamiento inicial del proyecto no descarta que en un futuro el sistema desarrollado pueda llegar a formar parte de uno más amplio.

2.6.2. Resumen de capacidades

En esta sección se definen las capacidades que deberá proporcionar el sistema una vez finalizado su desarrollo. La tabla 2.5 muestra una descripción más detallada de cada una de ellas. Las prioridades del proyecto se definen en el punto 2.7

[5] `http://www.fib.upc.edu/fib.html`

[6] `http://www.fib.upc.edu/fib/estudiar-enginyeria-informatica/enginyeries-pla-2003/normatives.html`

Necesidad	Prioridad	Impacto	Solución propuesta
Aprender siguiendo unos ejercicios propuestos según el progreso de cada usuario	Alta	Afecta a la calidad del aprendizaje. A mayor personalización, mejor aprendizaje.	Desarrollo de una herramienta que permita hacer un seguimiento exhaustivo del progreso de cada usuario para así poder proponer los ejercicios idóneos y la repetición espaciada en el tiempo más adecuada de los mismos.
Material a estudiar ampliable de forma sencilla	Media	Desarrollo de la aplicación. Percepción del usuario.	Desarrollar la herramienta de forma que sea fácilmente ampliable y modificable. Permitir que los usuarios puedan aportar ideas e incluso material de estudio.
Contenido de calidad	Media-Alta	Afecta a la calidad de la aplicación. Fácilmente detectable por los usuarios.	Realizar un control completo sobre todo el material que se introducirá en la aplicación. Controlar, además, las aportaciones de los usuarios y/o las modificaciones pertinentes. Comprobar con fuentes oficiales la veracidad de los materiales y su precisión.

Cuadro 2.4: Necesidades de los usuarios del sistema.

2.6.3. Suposiciones y dependencias

A continuación se enumeran las dependencias y supuestos en relación a este proyecto:

- Se necesita una conexión a Internet para poder utilizar el sistema.
- La aplicación se ejecutará en un navegador.
- La red podría dejar de funcionar. Esto, evidentemente, impediría la utilización de la aplicación.
- La aplicación está alojada en un servicio ofrecido por terceros. A pesar de que la red funcione, dicho servicio podría sufrir algún incidente y provocar así que el sistema no esté disponible.

2.6.4. Alternativas y competencia

Existen productos similares al presentado en este proyecto, ya existentes en el mercado y, por lo tanto, competencia para el sistema que se quiere desarrollar.

Una de ellas es *iknow*[7], la cual se centra, de momento, en idiomas tales como el japonés o el chino. Otra herramienta, planteada de forma diferente es *voxy*[8]. La idea de

[7]Más información en: `http://iknow.jp/` .
[8]Más información en: `http://voxy.com` .

Capacidad	Beneficios para el usuario	Características necesarias
Gestión de la interfaz	Poder utilizar mediante un navegador cualquiera la aplicación desarrollada.	Interfaz usable. Más concretamente: fácil de aprender, fácil de usar, robusta y flexible a los cambios.
Gestión de los ejercicios	Realizar ejercicios centrados en un concepto o ítem y de todos los aspectos relacionados al ítem.	Ejercicios cortos y concisos, centrados en poder examinar los aspectos de un ítem desde todos los puntos de vista posibles. Relacionado con el progreso del usuario.
Gestión del progreso	Poder consultar en todo momento su progreso a lo largo del tiempo con los diferentes ejercicios. Poder analizar dicho progreso para planificar adecuadamente los ejercicios venideros.	Análisis en tiempo real de los resultados del usuario mientras realiza los ejercicios. Aplicación de dicho análisis en la selección de ejercicios a realizar a continuación.
Gestión del material	Amplia variedad de materiales a estudiar. Poder disfrutar de diversas temáticas y de un abanico de ejercicios extenso para el estudio.	Controles exhaustivos para determinar que el material sea de calidad para proporcionarlo a los usuarios.
Gestión de los usuarios	Poder consultar su perfil según el dominio en la materia. Mostrar la evolución a través del tiempo de la relación de aciertos y fallos.	Definición de unos perfiles que determinen el grado de conocimiento de las diversas materias. Consulta de dicho conocimiento o progreso.
Gestión de la navegación	Facilidad de navegación por las diferentes partes de la aplicación a través de una `URL`.	Mostrar la información accedida según la `URL` escrita en la barra de navegación (formato `XML` o `JSON`).

Cuadro 2.5: Capacidades del sistema.

este sistema es integrar en la vida diaria del usuario un idioma, de momento tan sólo el inglés: de las noticias que lee o de algún documento que le interesa, el programa extrae pequeños ejercicios basados en esos textos que pertenecen al mundo real. También existe *Anki*[9] y *SuperMemo*[10], los cuales ofrecen estudio basado en tarjetas (o *flashcards*) que muestran una pregunta, el usuario piensa una respuesta y gira la tarjeta para comprobar si era correcta o no. Después ha de puntuar cómo de difícil le ha resultado el ejercicio (el primero es *Open Source*, el segundo no).

[9]Más información en: `http://ankisrs.net/` .
[10]Más información en: `http://www.supermemo.com/` .

2.6.5. Coste asociado al proyecto y licencias

Tal como se ha indicado previamente, este sistema se desarrollará a través de un proyecto de final de carrera, por lo que no habrá coste económico asociado. La aplicación se alojará y ejecutará en *Google App Engine*, el cual en un primer momento no es de pago hasta que la aplicación evolucione y sea necesario servir un número elevado de peticiones, tal y como se detalla en los puntos 6.3 y 6.3.1. Esto además permite evitar un gasto en infraestructura. El presupuesto para este proyecto se detalla en el punto 4.4.2.3.

Este proyecto se ha decidido distribuir bajo la licencia `GNU Affero GPL v3`[11], abreviadamente `AGPLv3`. Esta licencia permite la copia, modificación y distribución del código siempre que cualquier modificación se continúe distribuyendo bajo la misma licencia. Es una licencia que se centra en servicios que funcionan en la red. Además, como indica la idea del *strong copyleft*, obliga a publicar cualquier cambio de los servicios en red que utilizan código distribuido bajo una licencia `AGPLv3`, como es el caso de este proyecto.

Las licencias de los módulos de terceros, así como la licencia de este proyecto, no entran en conflicto en lo que a copia, modificación o redistribución se refiere. Todos los módulos que se han incluido en el directorio de trabajo de este proyecto tan sólo han sido utilizados, en ningún caso se ha modificado el código proporcionado. Los pasos necesarios para continuar el desarrollo se especifican en el anexo A.

Respecto el código `CSS`, el cual se ha desarrollado heredando de los ficheros proporcionados por *Bootstrap*, se ha escogido la licencia `Apache License, Version 2.0`.

En las tablas 2.6 y 2.7 se detallan los módulos de terceros que se han utilizado, los cuales son gratuitos, así como su licencia de distribución y su uso. Más adelante, en el apartado 6.6.3, se detallan las funcionalidades de cada uno de estos módulos.

Módulo	Licencia	Más información
Google App Engine SDK	`Apache License, Version 2.0`	`http://www.apache.org/licenses/LICENSE-2.0`
appengine-rest-server	`Apache License, Version 2.0`	`http://www.apache.org/licenses/LICENSE-2.0`
python-pagination	`GNU GPL, Version 2.0`	`http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`
python-dateutil	`PSF License, Version 1.5`	`http://docs.python.org/2/license.html`
gae-sessions	`Apache License, Version 2.0`	`http://www.apache.org/licenses/LICENSE-2.0`
PlayThru	`The MIT License`	`http://mit-license.org/`

Cuadro 2.6: Resumen de las licencias de terceros (implementación del sistema)

[11] Más información en: `https://www.gnu.org/licenses/agpl.html` .

Módulo	Licencia	Más información
Twitter Bootstrap	`Apache License, Version 2.0`	`http://www.apache.org/licenses/LICENSE-2.0`
Jinja2	`3-clause BSD license ("Modified BSD License")`	`http://opensource.org/licenses/BSD-3-Clause`
Less CSS	`Apache License, Version 2.0`	`http://www.apache.org/licenses/LICENSE-2.0`
jQuery	`Apache License, Version 2.0`	`http://www.apache.org/licenses/LICENSE-2.0`
javascript-form-validation	No especificada	El autor permite el uso de la siguiente manera: "*You may adapt this script for your own needs, provided these opening credit lines are kept intact.*"

Cuadro 2.7: Resumen de las licencias de terceros (interfaz *web*)

2.7. Orden y prioridades

Es necesario indicar la importancia relativa de las capacidades del proyecto definidas en el apartado 2.6.2 y que serán implementadas en las diversas fases del sistema.

El apartado 2.8.3 servirá para medir dicha importancia relativa y cómo ello cuadra con el plan de desarrollo de cada fase. Esta importancia se reflejará en la determinación de la prioridad de cada capacidad en relación a la iteración en particular en que se encuentre.

En concreto se han definido las siguientes etapas necesarias para el desarrollo del proyecto así como las capacidades a desarrollar incluidas en cada una de ellas:

- Fase Inicial
  a. Iteración I1
     - Documentación y preparación del entorno de trabajo.
     - Especificación inicial: objetivo, alcance, planificación, casos de uso, etc.
- Fase de Elaboración
  a. Iteración E1
     - Especificación detallada de los casos de uso: flujos principales y alternativos.
  b. Iteración E2
     - Diseño de la arquitectura del sistema.
     - Diseño de la arquitectura de la información.
     - Revisión de los casos de uso definidos en la iteración E1. Redacción de la versión final.

- Fase de Construcción

  a. Iteración C1
     - Gestión de progreso.
     - Gestión de interfaz.
     - Etapa de pruebas intermedia.

  b. Iteración C2
     - Gestión de ejercicios.
     - Gestión de navegación.
     - Gestión de usuarios.

  c. Iteración C3
     - Revisión del funcionamiento de los módulos implementados.
     - Etapa de pruebas intermedia.

  d. Iteración C4
     - Gestión de material.
     - Etapa de pruebas de final de fase. Definición del prototipo.

- Fase de Transición

  a. Iteración T1
     - Gestión y aplicación de posibles cambios.
     - Etapa de pruebas final. Definición de la versión final del producto.
     - Documentación del sistema. Ayuda al usuario.

Esta división de etapas dentro de cada una de las fases definidas por `RUP` se justifica de la siguiente manera:

- En la **primera fase** lo esencial es la documentación y la especificación. En este punto es vital adquirir el conocimiento necesario acerca de la metodología `RUP`, las herramientas que se necesitarán para la implementación y la adecuación del espacio de trabajo. También es necesario producir la primera documentación del proyecto, la cual se ha incluido en esta memoria. Esta documentación consta de la planificación inicial, el análisis de riesgos, la visión global y el alcance del proyecto, el desarrollo del documento de vocabulario, la definición de requisitos y de casos de uso, el presupuesto inicial y la asignación de recursos, el esquema inicial de desarrollo de software y, finalmente, el detalle de las actividades que se llevarán a cabo en el sistema. Como una correcta planificación es vital para la obtención de una buena solución final, cabe destacar que este es el motivo por el que esta fase es extensa.

- En la **segunda fase** se finaliza la documentación detallada de los casos de uso del sistema. La ventaja de utilizar una metodología de desarrollo iterativa es que permite observar con mucha más facilidad errores o cambios necesarios en el sistema. En este caso, una primera definición detallada de los casos de uso seguro se verá modificada en la segunda iteración de esta fase ya que, tras haber diseñado la arquitectura del sistema y la arquitectura de la información (cómo se estructurará el sistema para funcionar y cómo lo hará la información dentro de éste), es inevitable observar cambios necesarios en la primera definición de los mencionados casos de uso.

- La **tercera fase**, o fase de construcción, se centra en la implementación de la aplicación en su totalidad. Se ha dividido en cuatro iteraciones puesto que se ha creído necesario establecer un orden y, por tanto, una prioridad para alcanzar el objetivo de este proyecto. Como se puede observar, en primer lugar se implementa la gestión del progreso de cada usuario y la interfaz de la aplicación. Esto se ha decidido así ya que es la parte más importante del proyecto y es imprescindible que funcione. En estas etapas se desarrolla el algoritmo que calculará las repeticiones de los ejercicios y, por consiguiente, el progreso del usuario. La interfaz es necesaria para poder visualizar las pruebas que se realicen en la etapa de pruebas intermedia de forma cómoda. Se considera que, a pesar de ser al inicio de fase, éste es un punto de inflexión en el desarrollo puesto que unirá la parte más importante de la aplicación con la parte más visual de la misma. En la segunda iteración, se procede con el desarrollo de otra de las partes importantes y uno de los objetivos del proyecto: los ejercicios. Es un objetivo vital el que los ejercicios estén bien definidos para que sean agradables de realizar y den la sensación al usuario de que está invirtiendo su tiempo en una actividad valiosa para incrementar su conocimiento respecto a una materia. En esta misma iteración, también se implementa la gestión de navegación y de usuarios de la aplicación y en la siguiente iteración se comprueba la corrección de los módulos implementados hasta el momento y se realiza una primera etapa de pruebas. Finalmente, en la última iteración de esta fase, se realiza la implementación de la gestión de material, la cual vendrá seguida de una etapa de pruebas completa de la aplicación desarrollada hasta el momento y así se podrá definir la primera versión prototipo de la aplicación final.

- Para terminar, en la **cuarta fase**, o fase de transición, se gestionarán los posibles cambios que se hayan podido producir tras la etapa de pruebas de la iteración C4 para llegar al producto final. Sin embargo, antes de dar por finalizada esta fase, es necesario que esta gestión de cambios vaya seguida muy de cerca por una nueva batería de pruebas para verificar que dichos cambios mantienen la esencia de la aplicación y tan sólo han servido para mejorar errores. La etapa final será la de la documentación necesaria que falte por redactar para entregar con el proyecto. Además, será necesaria la confección de ayuda al usuario para la aplicación, la cual es necesario que se redacte en este punto ya que se requiere que la aplicación esté implementada por completo.

2.8. Análisis de requisitos

En este punto se analizan todos los requisitos que son necesarios para el desarrollo de la aplicación. Es imprescindible este análisis en la fase inicial del proyecto, ya que de esta manera se definen las necesidades de la aplicación, tanto funcionales como no funcionales, para así poder proceder a su diseño teniendo en cuenta estos requisitos.

2.8.1. Requisitos funcionales

Los requisitos funcionales definen el comportamiento interno del sistema. Se centran en el análisis de datos, cálculos y detalles técnicos que permitirán pasar de los casos de uso definidos en el apartado 5.3 a la realidad de una aplicación plenamente funcional. A continuación se enumeran los requisitos funcionales de la aplicación:

- El sistema ha de permitir a cualquier usuario con una cuenta de *Google* autenticarse en el sistema.
- El sistema ha de permitir al usuario que contacte con el responsable (técnico o gestor) de la aplicación para cualquier duda o comentario que desee realizar.
- El sistema ha de mostrar al usuario información acerca del funcionamiento de la aplicación (manual de usuario) para que pueda utilizarla por primera vez de forma fácil.
- El usuario se puede suscribir o desuscribir a cualquiera de los cursos disponibles tantas veces como desee.
- El sistema pone a disposición del usuario la posibilidad de estudiar de forma espaciada en el tiempo problemas distribuidos en series e incluidos en cursos a los que esté suscrito.
- El usuario puede consultar el progreso de estudio de un curso al que está suscrito.
- El sistema pone a disposición del usuario información de interés tal como los términos de uso de la aplicación, la política de privacidad, las preguntas realizadas frecuentemente o datos generales de cada curso.
- El sistema permite crear material de estudio a todo usuario registrado en el sistema. Esto incluye problemas, series de problemas y cursos.
- El sistema permite a un usuario con privilegios hacer que otro usuario cualquiera registrado en el sistema tenga privilegios de administrador (usuario gestor).

2.8.2. Requisitos no funcionales

En este punto se enumeran y analizan todos aquellos requisitos del sistema que no se contemplan en los casos de uso pero que también son de vital importancia para el buen funcionamiento del proyecto. Son necesarios para definir los aspectos imprescindibles para la puesta a punto del sistema. También se describen elementos orientativos que deben ser conocidos antes de definir los detalles del sistema para obtener así un producto final que satisfaga todos los requisitos planteados.

2.8.2.1. Ayuda y documentación

El proyecto ha de ir acompañado de una documentación clara y sencilla que abarque todo lo necesario para el mantenimiento y ampliación del mismo. Es necesario notar la diferencia entre la documentación orientada al usuario que puede continuar desarrollando el proyecto respecto de la del usuario final de la aplicación. En este sentido se han detallado a continuación dichas diferencias:

Documentación para el usuario desarrollador El usuario que quiera continuar el desarrollo de este proyecto tendrá a su disposición documentación que constará de los siguientes aspectos:

- Requisitos mínimos del sistema.

- Instrucciones de instalación.
- Documentación interna del código y de su estructura.
- Problemas conocidos y errores no resueltos, si los hubiera.
- Mejoras propuestas ordenadas según su importancia.

Documentación para el usuario final El usuario final de la aplicación es cualquier persona que desee utilizar la herramienta para aprender una materia disponible y/o desee crear material para el estudio. Por lo tanto, la ayuda de que dispondrá este tipo de usuario se define a continuación:

- Ayuda *online* entendida como tutoriales para el primer uso de la aplicación o guías de tipo *Frequently Asked Questions* (a partir de ahora `FAQ`); útil para resolver los problemas o las dudas más comunes que puedan surgir. Se puede encontrar en la sección *Help*[12].
- Formulario de contacto para poder realizar cualquier consulta en la sección *Contact*[13].
- Características del sistema definidas en la sección *About*[14].
- Un *tour* para cada uno de los cursos que hay disponibles que con sencillas capturas y explicaciones le darán una idea del funcionamiento de la aplicación cuando esté estudiando. Se puede consultar en la sección *Take the tour*[15] sin necesidad de que el usuario esté autenticado en el sistema, tan sólo consultando más información de un curso en concreto.

2.8.2.2. Componentes adquiridos

A priori, no hay necesidad de adquirir ningún tipo de componente, ya que todo el código necesario estará alojado en repositorios *online* como *BitBucket*[16] y la aplicación será accedida desde los servidores de *Google*.

2.8.2.3. Diseño modular

El sistema ha de ser flexible para poder añadir nuevo material de estudio así como para poder ampliar funcionalidades diversas.

2.8.2.4. Errores del sistema

Los errores deben quedar almacenados en un fichero de registro específico para la aplicación. En ellos se debe incluir un identificador del error, la descripción del motivo del error, el día y hora en que ocurrió y la clase en la que se generó. Es necesario contar, por tanto, con un sistema capaz de reaccionar apropiadamente ante cualquier imprevisto. Se minimizarán al máximo los errores de acceso al sistema a través de la red. Para solventar posibles pérdidas de datos, se realizarán las copias de seguridad adecuadas en cada caso.

[12] *Help*: `http://ulearnpfc.appspot.com/help` .
[13] *Contact*: `http://ulearnpfc.appspot.com/contact` .
[14] *About*: `http://ulearnpfc.appspot.com/about` .
[15] *Take the tour* (curso de inglés): `http://ulearnpfc.appspot.com/tour_english` .
[16] `https://bitbucket.org/`

2.8.2.5. Fiabilidad y disponibilidad

El sistema ha de estar disponible 24 horas al día, 7 días a la semana. En caso de ser necesario realizar tareas de mantenimiento del mismo, se efectuarían después de analizar las estadísticas de acceso al sistema y en los momentos de menor afluencia de usuarios, avisando con antelación si es posible.

2.8.2.6. Imagen corporativa

El logo del sistema aparece en la página principal accediendo a `http://ulearnpfc.appspot.com` de la misma forma que aparece en la documentación del proyecto en el anexo F. Debido a que se ha planteado como un proyecto final de carrera no se ha diseñado ningún plan de *marketing* o de promoción del producto final. No se descarta que en un futuro sí se intente comercializar.

2.8.2.7. Legislación vigente, copyright y otras advertencias

El sistema de información debe atenerse a la Ley Orgánica de Protección de Datos (15/1999) ya que se almacenarán datos personales de los usuarios; en concreto, la dirección de correo electrónico tal y como se detalla en el apartado 6.6.8.4.

2.8.2.8. Rendimiento: Capacidad

La utilización de bases de datos es imprescindible para el uso de la aplicación tal y como está planteada. Se estima que los accesos a la misma serán escalables de forma automática por *Google App Engine* tal y como se especifica en su página *web* [4].

2.8.2.9. Rendimiento: Tiempo de respuesta

Se asume que los accesos a las bases de datos que almacenan el material deben ser lo suficientemente rápidos para agilizar el proceso de resolución de ejercicios y, por tanto, hacer del estudio una tarea agradable. Es por este motivo que se ha determinado un máximo de dos segundos por cada petición que se realiza a la base de datos de *Google App Engine*.

2.8.2.10. Restricciones de diseño

La restricción más destacable del sistema es la dependencia de disponibilidad de los sistemas que proporcionan acceso a la herramienta, básicamente *Google App Engine*, ya que alojarán el código y permitirán el uso de la aplicación.

2.8.2.11. Soportes

El único requisito para el uso del sistema es disponer de un navegador. Ya que la aplicación será diseñada en un lenguaje de programación independiente del sistema operativo, no habrán problemas de compatibilidad en este sentido. Se realizarán las pruebas necesarias en las etapas correspondientes para detectar cualquier posible incompatibilidad con los navegadores más usados disponibles en el mercado.

2.8.2.12. Usabilidad y facilidad de uso

El usuario no necesitará tener unas aptitudes destacables en ningún aspecto concreto para poder hacer un uso normal de la aplicación, aparte de poseer conocimientos básicos para el manejo de un navegador *web*. Esto deberá ser probado para confirmar que la usabilidad es buena, demostrando que las interfaces son intuitivas y, por tanto, sencillas y agradables.

Adicionalmente, la *web* ha de mostrar al usuario información acerca de cómo ha llegado hasta la página que está viendo para que pueda cambiar el flujo de navegación, si lo desea ("hilo de Ariadna" o *breadcrumb* en inglés).

2.8.3. Otros requisitos del sistema

Estos son los factores de calidad que se tienen en cuenta a la hora de desarrollar el proyecto ordenados por importancia (de mayor a menor):

- **Corrección**: Las tareas serán realizadas con exactitud, cumpliendo su propósito para satisfacer al usuario.
- **Compatibilidad**: El proyecto debe cumplir los estándares actuales y de programación para facilitar compatibilidades entre plataformas y navegadores donde se ejecute.
- **Robustez**: Al ser utilizado vía Internet, es necesario contar con un sistema capaz de reaccionar apropiadamente ante cualquier imprevisto. Se minimizarán al máximo los errores de acceso al sistema a través de la red.
- **Funcionalidad**: El sistema ha de proporcionar una mejora en el autoaprendizaje, haciendo del mismo una actividad entretenida para así atraer el interés por su uso.
- **Extensibilidad**: El sistema debe ser capaz de adaptarse a cambios de especificación, ya que se puede contemplar la ampliación del mismo. Esto ya se tiene en cuenta con la gestión de material. Además, se ha tener en cuenta la actualización de ejercicios y posibles cambios de los mismos según las necesidades de los usuarios.
- **Mantenibilidad**: El sistema debe ser fácil de mantener y de añadir nuevos módulos o modificarlos.
- **Usabilidad**: La interfaz ha de ser lo suficientemente sencilla para que cualquier persona sea capaz de utilizarla.

2.9. Análisis de riesgos

En este apartado se presenta una lista de riesgos conocidos que pueden afectar al proyecto. Se detallan acciones para evitarlos o, en caso que esto no fuera posible, paliar sus efectos. El éxito o el fracaso de un proyecto depende de los riesgos asociados a éste, por lo que es necesario reconocer su existencia, clasificarlos según su impacto y preparar un plan de mitigación y, en el caso peor, de contingencia. El plan de mitigación de un riesgo normalmente conlleva un coste económico asociado mientras que el plan de contingencia se basa en aplicar las directivas definidas en el plan de mitigación para minimizar los efectos del riesgo que finalmente se ha producido.

2.9.1. Planteamiento inicial

Se ha de definir una estrategia de planificación para el proyecto, de forma que se pueda medir el nivel de garantía de éxito que tiene. Para conseguirlo es necesario definir tres conceptos:

- La **concepción**; para de esta forma adoptar una visión global, general y a la vez detallada del objetivo. En este caso, se trata de que el usuario pueda utilizar una herramienta que le permita aprender una materia de forma fácil y amigable. La diferencia respecto otras formas de aprendizaje es el espaciado en el tiempo de los ejercicios propuestos teniendo en cuenta que éstos, además, se propondrán según el progreso del usuario en dicha materia.

- La **estructuración**; prever los pasos, tareas y aspectos fundamentales de la concepción. Aplicado al proyecto actual sería la realización de una lista de etapas y tareas esenciales a desarrollar para conseguir el objetivo final. La asignación de prioridades a dichas tareas y la revisión de las mismas para discernir cuáles se pueden llegar a suprimir, o cuáles se pueden añadir, es esencial para definir la estructura.

- La **planificación**; para poder detallar el camino a seguir en el proyecto, relacionando tareas con recursos y duración. Por lo tanto, será necesario una planificación adecuada y la definición de dependencias e hitos.

Esta estrategia de la planificación, junto con la de la elaboración y con el análisis de los riesgos (tal y como se detalla en el apartado 4.2.1) harán que el proyecto sea viable y tenga éxito.

2.9.2. Clasificación y lista de riesgos

En la tabla 2.8 se muestra una clasificación por categorías de los riesgos que se contemplan en este proyecto.

Los dos riesgos más importantes a tratar en este proyecto son los siguientes: “La repetición espaciada en el tiempo de los ejercicios no se produce adecuadamente” y “El cálculo del progreso del usuario en una materia no es correcto”. Esto es así ya que el objetivo de este proyecto es que el usuario pueda aprender una materia haciendo uso de un algoritmo que, de manera espaciada en el tiempo, va proponiendo ejercicios según el progreso del usuario en dicha materia.

Familia	Riesgo
Usuario	Requisitos poco definidos y variables. Exceso de expectativas.
Calendario	Tiempo de desarrollo excesivo del proyecto.
Recursos	Pérdida de datos. Fallo del servidor donde se aloja la aplicación. Problemas con el entorno de trabajo y el lenguaje de programación. Desconocimiento de la metodología `RUP` y sus herramientas.
Experiencia de usuario	La repetición espaciada en el tiempo de los ejercicios no se produce adecuadamente. El cálculo del progreso del usuario en una materia no es correcto. Mal funcionamiento de la aplicación en los navegadores disponibles. Mala usabilidad de la aplicación.
Gestión	Aumento inesperado del coste del proyecto.

Cuadro 2.8: Clasificación según la tipología.

En las tablas 2.9 y 2.10 se muestra una nueva clasificación de los riesgos, esta vez según la probabilidad de que ocurran y del impacto que tendrían en el proyecto.

Probable	Improbable
Problemas con el entorno de trabajo y el lenguaje de programación. Mal funcionamiento de la aplicación en los navegadores disponibles.	Requisitos poco definidos y variables. Exceso de expectativas. Tiempo de desarrollo excesivo del proyecto. Pérdida de datos. Fallo del servidor donde se aloja la aplicación. La repetición espaciada en el tiempo de los ejercicios no se produce adecuadamente. El cálculo del progreso del usuario en una materia no es correcto.

Cuadro 2.9: Probabilidad de ocurrencia de los riesgos con un **impacto alto**.

Probable	Improbable
Desconocimiento de la metodología RUP y sus herramientas.	Mala usabilidad de la aplicación. Aumento inesperado del coste del proyecto.

Cuadro 2.10: Probabilidad de ocurrencia de los riesgos con un **impacto bajo**

A continuación se plantea la descripción completa de cada riesgo que incluye los planes de mitigación y contingencia, haciendo especial hincapié en los riesgos citados previamente.

2.9.2.1. Requisitos poco definidos y variables. Exceso de expectativas

Descripción Se espera de RUP que proporcione requisitos más estables y estimaciones más precisas que los modelos en cascada o iterativos tradicionales, pero no se le puede exigir que resuelva problemas inherentes al desarrollo de programas. Es preciso un nuevo planteamiento para dar más énfasis a la aceptación de cambios y no intentar estabilizar y congelar los requisitos, ya que esta idea no siempre es parte de la informática; es necesario comprender que no se puede prever de manera totalmente fidedigna las estimaciones necesarias en el proyecto, pero que se ha de ir mejorando con la experiencia y las iteraciones. Y para todo ello, es necesario entender bien lo que pide el cliente y ser dinámicos con los cambios: aceptar que los cambios ocurren, pero que no siempre son aceptables.

También es posible que haya poca o insuficiente comunicación con el cliente, debido a la indisponibilidad de éste o por suponer hechos y tomar decisiones sin tener suficientes datos. Este último factor puede verse agravado en fechas cercanas a entregas o en visitas del cliente. La falta de comunicación puede derivar en un exceso de expectativas por parte del cliente, el cual puede determinar que el producto final no cumple con sus esperanzas.

Impactos Este riesgo tiene un impacto muy grande en la calidad final del producto y en la satisfacción del cliente, y por ello obtiene tan alta puntuación en su magnitud. Ambas cosas repercuten en la reputación del desarrollador del proyecto. Es posible entregar un producto similar al que el cliente ha pedido, pero diferente en algunos aspectos clave, ya que no ha habido una correcta comunicación entre las partes. Hay que estar dispuesto a aceptar que los cambios ocurren y que es necesario que el cliente forme parte del proceso activamente.

Indicadores Un indicador útil es el número de citas concertadas con el cliente para dialogar acerca del proyecto y de ciertas decisiones que requieren de su visión y aprobación: será necesario realizar un control del número de reuniones con el cliente y escuchar de manera atenta a sus intervenciones y así detectar posibles errores en la comprensión de los requisitos. Hay que hacer notar que, en muchas ocasiones, el cliente no dispone de los mismos conocimientos tecnológicos que los desarrolladores, por lo que su visión puede ser muy diferente a la del equipo de desarrollo, sin que ello sea mejor o peor.

Estrategia de mitigación Resulta acertado determinar una cierta periodicidad en las reuniones con el cliente, en base a la duración estimada del proyecto, y disponer de otros medios de comunicación alternativos (correo electrónico, teléfonos de oficina y móviles, dirección postal, etc.).

Plan de contingencia En caso de detectar que la comunicación con el cliente es baja o insuficiente, es necesario ser capaz de reconocer que es necesario convocar una reunión con el cliente aunque no estuviera planificada (esto es, convocar reuniones siempre que sea necesario, de manera dinámica). Si los requisitos son ambiguos o difieren en lo que el cliente pide, la actuación deber ser idéntica. Será necesario determinar el origen del problema (falta de diálogo o de entendimiento, requisitos ambiguos u opuestos, falta de concreción o indecisión del cliente) y llegar a un acuerdo que satisfaga a ambas partes en cuanto sean detectados dichos problemas. También será necesario tener un plan de actuación diseñado que incluya pensar en las alternativas a las necesidades básicas del proyecto en caso de un cambio de opinión a última hora en requisitos clave.

2.9.2.2. Tiempo de desarrollo excesivo del proyecto

Descripción El tiempo planificado inicialmente para el desarrollo del proyecto es insuficiente ya que, o bien han surgido imprevistos que han retrasado la evolución, o bien el proyecto era de mayor envergadura de lo que se pensó en un primer momento.

Impacto El hito de finalización del proyecto se verá retrasado, con lo cual la planificación se verá modificada.

Indicadores Los principales indicadores para este riesgo provienen de la experiencia de los tres componentes clave del proyecto: del tutor, del director y de su autor. También puede ser un indicador una planificación carente de detalles donde se observen pocas tareas, o unos recursos ineficientemente asignados a las necesidades reales.

Estrategia de mitigación En general, es posible darse cuenta de este riesgo a través del tiempo que se le quiere dedicar al proyecto y a través de la realización de una planificación bien detallada y precisa. Una definición de los hitos bien marcada y un cumplimiento de los mismo de una forma estricta evitará que este riesgo ocurra.

Plan de contingencia Una vez que el riesgo haya sucedido, la manera de intentar contenerlo es recortar en las funcionalidades a implementar, haciendo que aquellas que no son imprescindibles para alcanzar el objetivo se vean relegadas a una segunda versión del producto, o incluso, a un proyecto distinto. Para ello es importante tener claro la priorización de tareas que se decidió en la planificación y replanificar los hitos que no sean imprescindibles, si es necesario.

2.9.2.3. Pérdida de datos

Descripción Pérdida de documentación, código o documentos internos; es decir, trabajo realizado por el desarrollador del proyecto.

Impactos El impacto se verá agravado según cada caso particular de pérdida de datos, en función de la cantidad de datos, la dificultad de su elaboración y el grado de importancia de los mismos. En cualquier caso, supone un retraso de mayor o menor gravedad en los plazos del proyecto, y es algo especialmente crítico en la relación con el cliente.

Indicadores Este riesgo sólo se puede detectar una vez haya ocurrido, ya que es una situación imprevisible. Es preciso, no obstante, un análisis del origen del problema, pero no es posible hacerlo antes de que ocurra, tan sólo intentar que no ocurra nunca.

Estrategia de mitigación El trabajo se ha planificado de manera que se usa un sistema de control de versiones en un servidor fiable y externo, el cual además realiza copias de seguridad diarias de estos datos como medida de protección adicional.

Plan de contingencia En caso de que este riesgo se materialice, se puede recuperar el documento gracias al control de versiones utilizado. El peor caso se presenta si se pierde la copia de trabajo local sin haber realizado una confirmación de los cambios, ya que es la que contiene las últimas modificaciones; en este caso, se habrá perdido una cantidad de horas equivalente a las horas desde la última vez que se enviaron los cambios al servidor central de versiones. En este caso, el impacto del riesgo es elevado; en cualquier otro caso, el impacto del riesgo es muy bajo, ya que ha sido previsto y prácticamente anulado en la estrategia de mitigación.

2.9.2.4. Fallo del servidor donde se aloja la aplicación

Descripción No es posible acceder a la aplicación ya que el servidor en el que se aloja no responde, o responde muy lentamente. Esto puede ser debido a diversos motivos, siendo los más comunes los problemas de conectividad con el exterior o con el proveedor de servicios de Internet, que el servidor está averiado (de mayor o menor gravedad) o bien que el sistema ha sido comprometido.

Impactos El mayor impacto es la pérdida de visitas a la aplicación y que se podría llegar a traducir en la no fidelización de los usuarios. Otro problema derivado del anterior es el daño a la reputación y a la confianza depositada por los visitantes en la aplicación.

Indicadores Es posible detectar la presencia de este tipo de malfuncionamiento a través de programas específicos para tal fin, que tendrán que ser instalados, configurados y mantenidos por los administradores del sistema informático.

Estrategia de mitigación Hay que tener presente que es imposible que este riesgo no se materialice en algún momento, por lo que sólo se puede reducir el número de apariciones (de ahí la elevada importancia). Se pueden adoptar medidas preventivas como la contratación de mantenimiento permanente y de respuesta rápida, así como crear un sistema redundante que minimice el tiempo de solución.

Plan de contingencia En caso de que este riesgo se materialice, se debe realizar un análisis del problema (conectividad, avería, ataques...) y actuar de una manera u otra según el resultado. En el primer caso, se deberá resolver el problema de conectividad (que

puede ser una avería externa); en los otros dos, el primer paso sería poner en producción el equipamiento de reserva, si existiera, y proceder a reparar el equipo que no funciona. Sin embargo, siempre se ha de poder informar al usuario de que existe un problema en el sistema y que se encuentra en proceso de resolución.

2.9.2.5. Problemas con el entorno de trabajo y el lenguaje de programación

Descripción Existe la posibilidad de que el lenguaje de programación no sea el adecuado y su funcionamiento no sea el esperado o, en otros casos, que su desarrollo no sea tan rápido como se esperaba. También es posible que el entorno de trabajo dé problemas durante el desarrollo, normalmente incompatibilidades.

Impacto El mayor impacto de este riesgo es la planificación, ya que si el entorno de trabajo no es el adecuado, el plazo de entrega puede verse ampliado de manera significativa.

Indicadores Este riesgo puede detectarse mediante la ayuda de los *diagramas de Gantt* que se realizan en cada fase y se deben poner los medios necesarios para solventar los problemas en el menor tiempo posible.

Estrategia de mitigación Es importante dedicar esfuerzos al aprendizaje del lenguaje de programación para prevenir errores básicos, pero esto no siempre es efectivo. Otra buena acción para mitigar el riesgo es realizar unas pruebas básicas en el entorno de trabajo para comprobar que todo funciona correctamente; por ejemplo, crear un pequeño proyecto nuevo, programar algo básico, probarlo y enviarlo al servidor de control de versiones; comprobando así que no hay problemas con el lenguaje, que el control de versiones funciona bien, etc.

Plan de contingencia Si se detectan problemas en plena fase de construcción, es necesario contactar con un experto en dicho lenguaje, utilizar los foros de programación y buscar soluciones alternativas.

2.9.2.6. Desconocimiento de la metodología `RUP` y sus herramientas

Descripción El no estar habituado al uso de `RUP` puede derivar en un producto final de menor calidad, realizado de manera poco eficaz.

Impactos La consecuencia principal que puede derivar de este riesgo es el retraso en la entrega de uno o más documentos, debido a un largo proceso de aprendizaje y familiarización con las herramientas y la metodología `RUP`.

Indicadores Este riesgo puede detectarse mediante la ayuda de los *diagramas de Gantt* que se realizan en cada fase. En caso de existir retrasos en las primeras tareas a realizar, una lectura detallada de estos diagramas podrá determinar si se deben a problemas de aprendizaje o de soltura con las nuevas herramientas y metodología a usar.

Estrategia de mitigación La fase inicial ha permitido iniciar el aprendizaje de las herramientas necesarias para el correcto desarrollo de este proyecto, por lo que el riesgo queda prácticamente anulado de cara a las fases finales. Se puede encontrar una breve visión de las herramientas utilizadas en este proyecto en el anexo D.

Plan de contingencia En caso de que este riesgo se materialice, se dedicarán un número determinado de horas a entender su uso. En cualquier caso, la planificación del proyecto ya ha previsto este riesgo, por lo que su coste y efectos han sido asumidos en gran parte.

2.9.2.7. La repetición espaciada en el tiempo de los ejercicios no se produce adecuadamente

Descripción Una de las dos características diferenciadoras de este proyecto no funciona adecuadamente: los ejercicios que se han de repetir en el tiempo según el progreso en una materia lo hacen de forma incorrecta. Este riesgo está estrechamente relacionado con el riesgo "El cálculo del progreso del usuario en una materia no es correcto".

Impactos Precisamente con este riesgo el objetivo del proyecto se ve amenazado de que no se cumpla. Es por esto que se le ha asignado un impacto muy alto.

Indicadores Esto se podrá detectar en la fase de pruebas intermedia que se producirá en la iteración C2 de la fase de construcción. En este punto es crucial comprobar que los ejercicios se repiten en el tiempo de forma adecuada.

Estrategia de mitigación Como se ha comentado en la parte de los indicadores, la manera de mitigar el riesgo es realizar una buena etapa de pruebas intermedia en la fase de construcción. Tan sólo así se podrá decir que se ha mitigado el riesgo. Como está relacionado con el riesgo previamente mencionado, para poder avanzar antes de implementar la gestión del progreso se asignarán unos valores aleatorios de progreso para determinar que la repetición de los ejercicios es correcta.

Plan de contingencia Si se avanza en la implementación y el riesgo se acaba produciendo se detectará en la siguiente fase de pruebas, una vez que se haya implementado la parte del progreso del usuario y se haya podido relacionar con todo lo anteriormente desarrollado. En este punto será necesario poner remedio a los problemas que se encuentren.

2.9.2.8. El cálculo del progreso del usuario en una materia no es correcto

Descripción Se trata de la segunda característica diferenciadora del proyecto. Como se ha comentado en el riesgo anterior, el cálculo del progreso va estrechamente relacionado con que los ejercicios se repitan de forma adecuada. Si este cálculo no se realiza correctamente, esa repetición también será incorrecta.

Impactos El impacto es elevado pues, igual que antes, el objetivo se ve amenazado de no cumplirse. También es evidente que la planificación se verá retrasada si hay problemas con este riesgo y se acaba materializando, puesto que no es una parte de la cual se puede recortar en funcionalidades para seguir avanzando.

Indicadores Similarmente al riesgo anterior, los indicadores serán las etapas de pruebas para determinar que todo funciona correctamente.

Estrategia de mitigación Para mitigar que este riesgo se produzca es necesario una completa etapa de pruebas antes de pasar a la siguiente iteración. Tal y como está planificado en el apartado 4.2.3, esta etapa tendrá lugar justo después de la gestión de material, en la iteración C4 de la fase de construcción. En ese punto se procederá a desarrollar el primer prototipo de la aplicación.

Plan de contingencia Si el riesgo se acaba materializando incluso después de dar por finalizada la etapa de pruebas de la iteración C4, será entonces detectado en la fase final, la de transición, cuando se estén gestionando posibles cambios ya que el prototipo de la iteración C4 será probado por personas ajenas al proyecto que puedan aportar mejoras en esta fase final. En este punto sí es crucial detectar que en la etapa de pruebas de la iteración C4 no se realizó una correcta programación del cálculo del progreso y por tanto se priorizará solucionar este problema.

2.9.2.9. Mal funcionamiento de la aplicación en los navegadores disponibles

Descripción Actualmente existen infinidad de navegadores disponibles para el usuario final, y no todos interpretan de igual manera las páginas web. Es muy difícil conseguir que todos muestren de la misma manera una determinada página web, ya que el problema son las pequeñas variaciones en el motor de interpretación de `HTML` y de `CSS`. Algunas de esas diferencias son tan importantes que pueden hacer que ciertas partes no funcionen correctamente o que, directamente, no sean visibles para el usuario final. Además, algunos usuarios pueden tener configuraciones incorrectas en su terminal que lleven a los mismos errores, aunque con distinto origen.

Impactos Dado que este riesgo está relacionado con el riesgo "Fallo del servidor donde se aloja la aplicación", los impactos son los mismos: pérdida de visitas en la aplicación y en la no fidelización de los usuarios, así como daño a la reputación por no poder ofrecer al usuario el servicio que espera.

Indicadores Durante la fase de implementación se pueden descubrir algunos problemas esenciales, pero será difícil poder decir que se ha cubierto la totalidad de los mismos.

Estrategia de mitigación Con la actual coyuntura tecnológica, este riesgo va a existir durante todo el proyecto. La mejor manera de reducir su impacto es tenerlo siempre presente, en especial durante la fase de elaboración, e intentar realizar un diseño que cumpla con los estándares actuales de la industria y que haya pasado el mayor número de pruebas posibles. Es tal la dificultad, que se puede estar satisfecho si se llega a un porcentaje mayor a un 95 %, que se conseguiría con una correcta visualización en los cinco navegadores mayoritarios. Otra medida es restringir el soporte ofrecido a un cierto subconjunto de navegadores, en los que se sabe que funciona todo correctamente e informar así al usuario.

Plan de contingencia En caso de que este riesgo se materialice, poco se puede hacer excepto solventar el fallo y corregirlo en la versión de producción tan pronto como sea posible. Se pueden emitir avisos para los usuarios cuya incorrecta configuración provoque que el acceso a la aplicación pueda no ser correcto, pero no se puede asegurar ni su solución ni que sea adecuadamente interpretado por el usuario final. En caso de errores de terceros, sería recomendable contactar con sus responsables para hacerles partícipes del error e intentar que lo solventaran. También es adecuado disponer de una manera de contacto visible para que, en caso de error, los usuarios puedan reportarlo fácilmente a los gestores.

2.9.2.10. Mala usabilidad de la aplicación

Descripción La interfaz para utilizar la aplicación no es agradable al uso.

Impactos Descontento en los usuarios, principalmente. Esto puede suponer la no fidelización de los usuarios con la aplicación, igual que una mala reputación en un primer momento.

Indicadores En las etapas de pruebas se puede comprobar que el tiempo que pasa el usuario con la aplicación es bajo, ya que al poco de haber estado usándola se siente cansado o con pocas ganas de continuar haciendo ejercicios.

Estrategia de mitigación Este riesgo se puede mitigar en la iteración C3, en la etapa de pruebas intermedia tras haber gestionado la navegación en la iteración C2 y la interfaz en la C1. Es aquí donde, de la misma forma que en riesgos anteriores, es necesario realizar un buen conjunto de pruebas que determinen el nivel de usabilidad de la aplicación y realizar en este momento los cambios necesarios.

Plan de contingencia Si finalmente se acaba produciendo el riesgo una vez que la aplicación se encuentre en producción, la mejor manera de solucionar problemas de usabilidad es permitir a los usuarios que de una forma fácil se puedan poner en contacto con los administradores y reportar aquellos errores que detecten. Una vez detectado el error, la estrategia seria la de priorizar su solución y avisar al usuario conforme está solucionado.

2.9.2.11. Aumento inesperado del coste del proyecto

Descripción El presupuesto de la planificación inicial no cuadra con lo que finalmente costará el proyecto.

Impactos Puede retrasar la planificación del proyecto debido a que hasta que no se consiga financiación no se pueda seguir desarrollando. También puede repercutir en etapas no implementadas debido a la falta de recursos.

Indicadores Los hitos financieros y una gestión de los recursos de forma eficiente deben poder indicar si el presupuesto no se ajusta a lo inicialmente planificado y pactado.

Estrategia de mitigación Como se ha comentado en los indicadores, la gestión financiera ha de ser al detalle y estar bien gestionada. Hay que ser flexibles en cuanto a gestión de recursos según las necesidades del proyecto si, por ejemplo, los requisitos se ven modificados a medida que se va desarrollando. Así mismo, también es necesario saber recortar en aquello que no sea clave para la obtención de un buen producto final. Una buena manera de mitigar el riesgo es establecer un presupuesto de contingencia en caso de imprevistos que represente un 5-10 % del total e informar del mismo al cliente.

Plan de contingencia En caso de que finalmente ocurra, será necesario actuar en consecuencia con ese 5-10 % del presupuesto para intentar no recortar en funcionalidades y seguir con lo establecido en la planificación. Se necesitarán recursos extra a tal efecto. Es probable que, en el caso peor y a pesar de este margen de maniobra, sea necesario limitar las funcionalidades a desarrollar.

2.9.3. Cálculo del impacto de los riesgos

Una vez identificados los riesgos, el siguiente paso es determinar su impacto. Lo que se espera obtener en este apartado es especificar, de manera más concreta, el impacto real que tendrían los riesgos a partir de las correspondencias definidas en las tablas 2.11 y 2.12. Para ello se analizará el impacto, definido como la probabilidad de ocurrencia del riesgo multiplicado por la estimación de pérdida. Por ejemplo, si existe un 25 % de probabilidad de que ocurra un riesgo que se estima retrasará el proyecto en 4 semanas, entonces el impacto de este riesgo es de 0,25 x 4 = 1 semana. Tanto la estimación de la probabilidad de ocurrencia como de la pérdida en tiempo se realizan de forma subjetiva, acotando la subjetividad tanto como ha sido posible.

Clasificación	Gravedad
1	Leve
2	Relevante
3	Importante
4	Grave
5	Crítico

Cuadro 2.11: Clasificación según la gravedad.

Porcentaje	Probabilidad de ocurrencia
$\leq 20\,\%$	Muy baja
$\leq 40\,\%$	Baja
$\leq 60\,\%$	Media
$\leq 80\,\%$	Alta
$> 80\,\%$	Muy alta

Cuadro 2.12: Clasificación según la probabilidad de ocurrencia.

Finalmente, pero no por ello menos importante, se priorizan los riesgos de manera que se sepa dónde centrar el esfuerzo de la gestión de dichos riesgos. El resultado final

se muestra en la tabla 2.13. Tan sólo se ha realizado una ordenación por probabilidad de ocurrencia. Se muestran, por tanto, todos los riesgos, ya que así es posible priorizar algunos de aquellos que producirían alguna pérdida muy grande.

Riesgo	Probabilidad	Estimación de la pérdida	Impacto real	Gravedad
Problemas con el entorno de trabajo y el lenguaje de programación	85 %	6 semanas	5 semanas	4
Mal funcionamiento de la aplicación en los navegadores disponibles	80 %	4 semanas	3 semanas	3
Desconocimiento de la metodología `RUP` y sus herramientas	60 %	2 semanas	1 semanas	2
La repetición espaciada en el tiempo no se produce adecuadamente	55 %	8 semanas	4 semanas	5
El cálculo del progreso del usuario en una materia no es correcto	55 %	8 semanas	4 semana	5
Pérdida de datos	35 %	3 semanas	1 semana	3
Tiempo de desarrollo excesivo del proyecto	35 %	6 semanas	2 semanas	4
Fallo del servidor donde se aloja la aplicación	30 %	1 semanas	$\frac{1}{2}$semana	3
Requisitos poco definidos y variables. Exceso de expectativas	20 %	5 semanas	1 semana	4
Mala usabilidad de la aplicación	15 %	3 semanas	$\frac{1}{2}$semana	2
Aumento inesperado del coste del proyecto	10 %	2 semanas	$\frac{1}{2}$semana	1

Cuadro 2.13: Resumen del cálculo del impacto de los riesgos.

2.10. Convenciones utilizadas

En este punto se detallan las convenciones tipográficas que se utilizan en este documento:

- Las expresiones resaltadas o importantes siempre se presentan con un estilo de **fuente en negrita**.

- Las siglas se presentan con una fuente al estilo de la que se usaba en las máquinas de escribir. Por ejemplo, al hablar de la metodología `RUP`.
- Las palabras o expresiones en otro idioma son mostradas en cursiva, como al hablar de un *software* o de contenido *online*.
- Las direcciones *web* que aparecen en este documento enlazan directamente con el recurso que representan en la red si se hace clic en ellas: `http://www.fib.upc.edu`.
- Finalmente, todo el código fuente, independientemente del lenguaje de programación en el que se muestre, ha recibido un formato específico utilizando el modo *verbatim*.

Capítulo 3

Arquitectura del sistema

En este capítulo se presenta la arquitectura del sistema de forma general, describiendo qué se espera de él y cómo ha de comportarse. Se definen las tareas que se pueden llevar a cabo, quiénes son los encargados de realizarlas y también se define la estructura y el propósito del sistema. En este punto es cuando se analiza la relación entre el análisis de *software* y el resto del proyecto, además de definir las capacidades y mecanismos básicos del sistema [2]. Para una visión más detallada de la arquitectura del sistema consultar el capítulo 5.

3.1. Objetivos de la arquitectura y restricciones del sistema

La arquitectura del sistema, dada su estructura, tiene una serie de objetivos a alcanzar y algunas restricciones que cumplir:

- Algunas de las restricciones del sistema vienen impuestas por el servicio *Google App Engine* ofrecido por *Google*, tal y como se detalla en el apartado 6.3.1.
- El sistema ha de estar siempre en funcionamiento y procesar todos los accesos de usuarios a la aplicación.
- El desarrollo del sistema ha de seguir la planificación detallada en el capítulo 4. Esta planificación pertenece a la fase inicial del proyecto.
- El sistema ha de cumplir con todos los requisitos detallados en el apartado 2.8.

3.2. Descripción de la arquitectura

El patrón arquitectónico escogido para la implementación del sistema es el definido en el paradigma Modelo-Vista-Controlador (`MVC`). En este patrón se separan en tres componentes distintos el código que proporciona la lógica del sistema o **controlador**, el código para almacenar y procesar (cuando sea necesario) los datos de la aplicación o **modelo** y el código para dar formato y poder mostrar la interfaz de usuario o **vista**.

De esta manera, es más sencillo organizar correctamente el código en el proyecto y poder ampliar dicho código de forma eficaz ante nuevos requisitos.

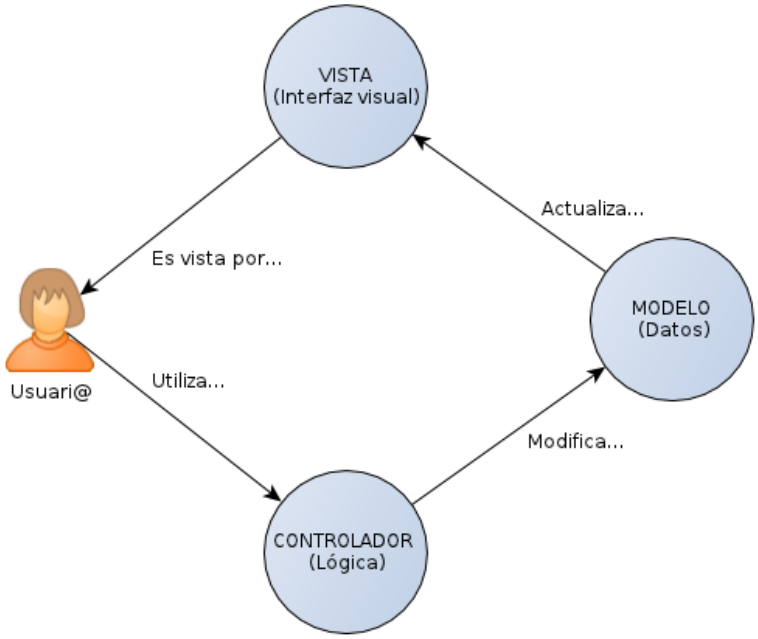


Figura 3.1: Esquema Modelo-Vista-Controlador.

A continuación se detalla cada uno de estos componentes del paradigma `MVC`.

3.2.1. Modelo

El código que define y gestiona el modelo de datos se ha implementado en un paquete denominado `db_ulearn`. Las clases necesarias son las siguientes:

- `Item`: define el objeto ítem o problema de una materia. Representa la unidad mínima de estudio para el usuario. Los ítems se agrupan en series.
- `Series`: define el objeto serie del sistema. Una serie es la unidad mínima para una misma sesión de estudio en un momento determinado para un usuario. Las series se agrupan en cursos.
- `Course`: define el objeto curso que puede ser estudiado. Un curso engloba series de problemas similares con el objetivo de alcanzar un mayor conocimiento sobre un determinado tema.
- `UserUlearn`: define el objeto creado a partir la autenticación (por primera vez) en la aplicación por parte de un usuario. Relaciona la información de los cursos a los que el usuario está suscrito así como el rol que tiene en el sistema (ver apartado 6.6.8.4).
- `ItemsBeingSolved`: define el objeto que se refiere a un ítem que está en proceso de resolución. Tan sólo se crean objetos en el breve período de tiempo en el que se están resolviendo problemas de una serie.
- `ProblemSolved`: define el objeto que se refiere a un problema finalizado adecuadamente por el usuario. Tan sólo se define un objeto por cada ítem, usuario y curso.

- `SeriesFinished`: define el objeto que se refiere a una serie finalizada por usuario y curso. Esta clase es útil para determinar si una serie ha sido resuelta con anterioridad o no por el usuario, para así poder determinar más fácilmente el orden de los problemas a resolver la próxima vez que inicie un serie de estudio

Todas las clases incluyen el código necesario para tratar los objetos definidos en cada una de ellas, ya sea a la hora de crearlos, modificarlos o eliminarlos.

3.2.2. Vista

El código que define la interfaz de usuario se ha implementado en el directorio *templates*. En él se han definido las plantillas utilizadas para dar forma a la interfaz, estableciendo unas pautas a partir de las cuales se genera el código `HTML` que se muestra al usuario.

3.2.3. Controlador

El código que enlaza la vista y el modelo se encuentra en el directorio raíz de la aplicación. El controlador proporciona la lógica del sistema y es, en definitiva, el código que se ejecuta al recibir las peticiones `HTTP` y modifica convenientemente los datos almacenados.

3.3. Introducción al algoritmo de repetición espaciada en el tiempo

La toma de decisiones en este proyecto está condicionada fuertemente por el algoritmo de repetición espaciada que permitirá planificar adecuadamente los problemas que tendrá que resolver el usuario. Es por este motivo que una visión a alto nivel de este algoritmo es necesaria en este punto de la fase inicial. En la figura 3.2 se muestra un esquema de este algoritmo. Más adelante, en el apartado 6.7.5 se detalla cada una de las clases y atributos que entran en juego.

3.4. Arquitectura desde el punto de vista de los casos de uso

En este apartado se proporciona una visión de la arquitectura del sistema desde el punto de vista de los casos de uso; se detallan los casos de uso que representan la funcionalidad del sistema, a muy alto nivel y sin un gran nivel de concreción. Esta visión global se detallará en apartados posteriores. Los casos de uso en negrita se consideran más importantes que el resto y se les ha asignado un número entre paréntesis que representa la iteración de la fase de construcción (es importante recordar que se han planificado un total de cuatro iteraciones en dicha fase) en la que se han de implementar:

Gestión de interfaz

- **consultarPerfil (1)**: Consulta del perfil del usuario en la aplicación. Se corresponde con la sección *My Desk*.

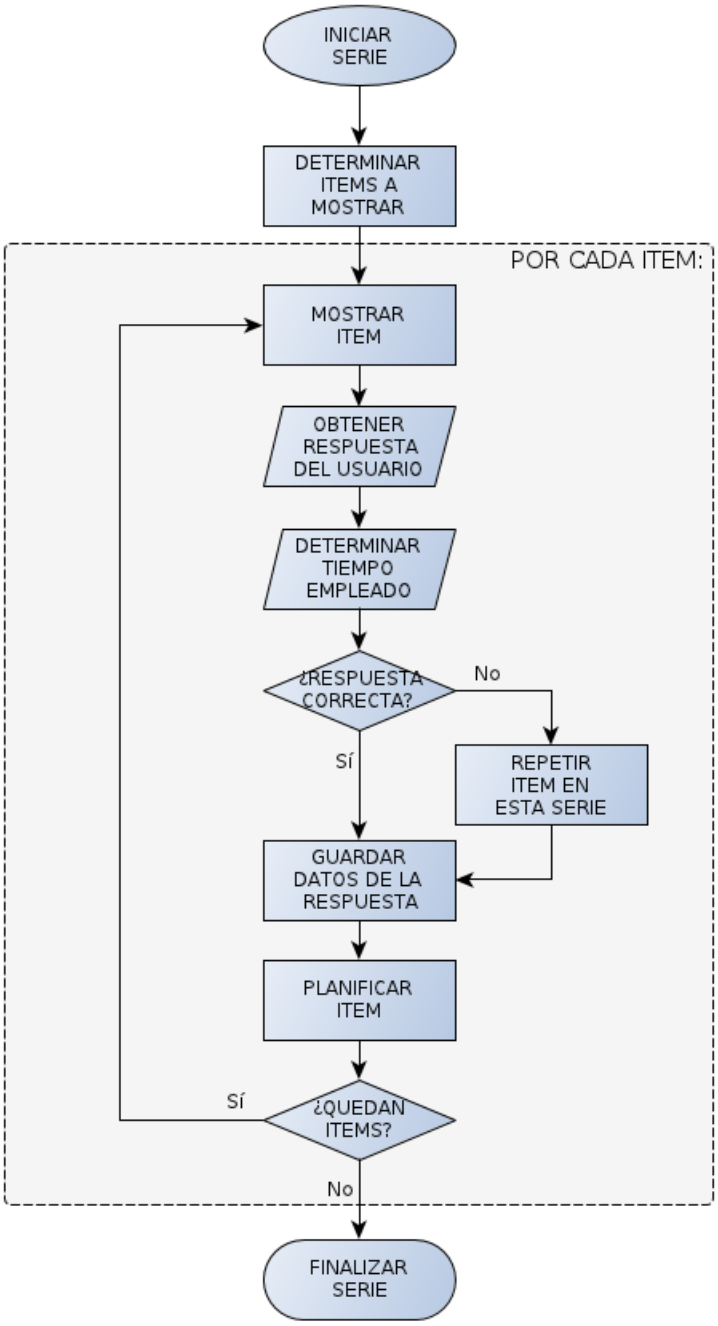


Figura 3.2: Flujo del Algoritmo de repetición espaciada.

- **consultarCursoEnEstudio (1)**: Consulta de los cursos a los que el usuario se ha suscrito. Se corresponde con la sección *My Desk*.
- **consultarCurso (1)**: Consulta de los cursos disponibles en el sistema.
- consultaVisitaGuiada (1): Realización de un pequeño *tour* por las diferentes secciones y servicios que ofrece la aplicación.
- consultaCaracterísticasULearn (1): Consulta de información general acerca del funcionamiento del sistema, útil para la primera toma de contacto entre el usuario y la

aplicación.

- consultaVisualizarProgresoEjemplo (1): Realización de un pequeño *tour* por el proceso de aprendizaje y qué ofrece la aplicación en el momento de la resolución de ejercicios.
- consultaQuienesSomos (1): Consulta de información acerca del equipo de desarrollo del sistema y de las motivaciones que han llevado a su desarrollo.
- **contacto (1)**: Mostrar un formulario de contacto a disposición de los usuarios.

Gestión de problemas

- **crearProblema (2)**: Creación de un nuevo problema en el sistema, por parte de un usuario con privilegios.
- **crearSerie (2)**: Creación de una nueva serie en el sistema, por parte de un usuario con privilegios.
- **escogerCursoAEstudiar (2)**: Selección por parte del usuario de un curso, de los disponibles en el sistema, para su estudio.
- **resolverSerie (2)**: Inicio de una sesión de estudio por parte del usuario.

Gestión de navegación

- **consultarProblema (2)**: Consulta de la información, por parte de un usuario con privilegios en el sistema, asociada a un problema.
- **consultarSeries (2)**: Consulta de la información asociada a una serie de problemas.
- **consultarAspecto (2)**: Consulta de la información, por parte de un usuario con privilegios en el sistema, asociada a un aspecto de un problema.
- consultarRol (2): Consulta de la información, por parte de un usuario con privilegios en el sistema, asociada a un rol determinado.
- consultarUsuario (2): Consulta de la información, por parte de un usuario con privilegios en el sistema, asociada a un usuario del sistema.

Gestión de usuarios

- **altaUsuario (2)**: Realización del alta de un usuario en el sistema.
- **confirmarAltaUsuario (2)**: Envío de un correo electrónico al usuario para que confirme el alta en el sistema.
- **bajaUsuario (2)**: Realización de la baja de un usuario en el sistema.
- bajaUsuarioTécnico (2): Realización de la baja de un usuario en el sistema por parte de otro usuario con privilegios.
- **loginUsuario (2)**: Inicio de sesión en el sistema por parte de un usuario.
- modificarPerfil (2): Cambio de la información mostrada al usuario en el apartado *My Desk*.

Gestión de progreso

- **consultarProgreso (1)**: Consulta del progreso en el aprendizaje por parte del usuario. Dicha información se puede consultar en la sección *MyDesk*.
- **consultarResultadoSerie (1)**: Consulta del resultado obtenido tras una sesión de estudio.
- **marcarProblema (1)**: Marcado automático de un problema para ser repetido tras un período de tiempo (porque ha sido fallado por el usuario).
- **actualizarProgreso (1)**: Actualización de la información asociada a los problemas que ha resuelto el usuario.
- actualizarHabilidades (1): Actualización de la información asociada a las habilidades del usuario. Se calcula respecto los problemas resueltos.
- actualizarLogros (1): Actualización de la información asociada a los logros del usuario. Se calcula respecto los problemas resueltos.

Gestión de cursos

- **crearCurso (4)**: Creación de un nuevo curso en el sistema, por parte de un usuario con privilegios.
- recepciónNuevoMaterial (4): Recepción de material susceptible de ser añadido como nuevo curso para el estudio.
- moderarNuevoMaterial (4): Moderación de material recibido, por parte de un usuario con privilegios.
- bajaMaterial (4): Baja del sistema de material de estudio, por parte de un usuario con privilegios.
- modificarCurso (4): Modificación de la información asociada a un curso, por parte de un usuario con privilegios.

Capítulo 4

Planificación del proyecto

Es necesario definir el esquema de desarrollo del proyecto de forma que proporcione toda la información necesaria para gestionarlo y planificarlo correctamente. El propósito de este capítulo es recopilar toda la información que va a ser necesaria para controlar el proyecto, gracias a los capítulos previos. Describe cómo se va a llevar a cabo el proyecto y es vital para una correcta gestión de las tareas a realizar por los desarrolladores del proyecto.

Este proyecto se inició el lunes día 28 de noviembre del 2011.

4.1. Planificación

4.1.1. Claves para una correcta interpretación

El objetivo principal de la fase inicial es describir el contexto del sistema, el modelo de negocio y trazar las bases del modelo de casos de uso del sistema. También es necesario definir de una forma adecuada el objetivo del proyecto y el alcance del mismo.

El uso de diagramas de *Gantt* es imprescindible, ya que ofrecen una visión rápida y global de las tareas a realizar, sus dependencias respecto otras tareas y la estimación de su duración.

Para la correcta interpretación de los diagramas que se utilizarán es necesario considerar que:

- Las etapas agrupan tareas.
- La tarea es la división más pequeña del proyecto e indica un trabajo a realizar. A la tarea se le asignarán unas horas de realización y unos recursos.
- Las etapas y las fases `RUP` están resaltadas en negrita en el diagrama.
- Los hitos no tienen carga de trabajo temporal.

4.1.2. Recursos de personal

Tal y como se explica en el apartado 2.5.3, el desarrollo del proyecto recae en una única persona, quien tendrá los perfiles de jefe de proyecto, analista y arquitecto.

4.1.3. Recursos informáticos

Los recursos necesarios son:

- Un ordenador con las herramientas de trabajo necesarias.
- Conexión a Internet.

4.1.4. Casos de uso

Durante esta iteración única de la fase inicial se identificarán a todos los actores y todos los casos de uso del sistema. Los cursos principales y alternativos más importantes se detallarán en el apartado 5.3.

Su diseño y definición a alto nivel se iniciarán en la próxima fase (elaboración), que consta de dos iteraciones (E1 y E2). Concretamente, se ha destinado parte de la primera iteración de esta fase a esta tarea.

4.1.5. Criterios de evaluación

El objetivo principal de esta iteración es definir el sistema con el nivel de detalle necesario para poder emitir un juicio objetivo sobre su viabilidad, desde un punto de vista puramente empresarial. En este juicio será necesario tener en cuenta los siguientes aspectos: funcionalidades, respuesta del sistema, capacidades, usabilidad, utilidad, etc.

Al finalizar esta iteración y fase se pondrá de manifiesto si es viable o no su realización.

Cualquier documento entregado durante la iteración será revisado por el responsable de los servicios informáticos, facilitando la integración y coherencia del proyecto.

4.2. Visión general del proyecto

4.2.1. Propósito, alcance y objetivos

En este documento se describe cómo se va a llevar a cabo el desarrollo de los servicios y el sistema de información **ULearn**. Los usuarios de este sistema dispondrán de una herramienta sencilla y útil para el aprendizaje de los cursos disponibles, tal y como se explica en los capítulos 2 y 3.

El desarrollo de este proyecto se basa en los requisitos detallados en el capítulo 2, y en la planificación temporal definida en el anexo B de esta fase de iteración única. Para realizar un resumen de todos estos documentos, se podría decir que la lista de pasos a dar en la planificación sería:

- Tener claro los objetivos.
- Centrarse en los requisitos del proyecto.
- Definir las expectativas a cumplir.
- Realizar un borrador de las tareas que se han de llevar a cabo.
- Analizar dicha lista observando objetivos y requisitos: añadir las tareas necesarias y eliminar las que no lo sean.

- Priorizar tareas.
- Realizar la planificación con la herramienta adecuada.
- Definir los recursos necesarios para ejecutar las tareas.
- Definir hitos y dependencias.
- Tener presente el calendario y la duración de cada tarea.
- Definir de forma definitiva los casos de uso del sistema.

En definitiva, es necesario definir una estrategia a seguir para poder desarrollar adecuadamente el proyecto. Una vez definida la estrategia de la planificación (ver apartado 2.9.1), la de la fase de elaboración se podría resumir de la siguiente manera:

- Una vez finalizada la implementación de la aplicación, realizar pruebas completas antes de la definición de un prototipo viable, o versión beta.
- Pruebas de la aplicación por personas ajenas al desarrollo.
- Detección de errores y gestión de cambios.
- Definición de la versión final de la aplicación.

4.2.2. Restricciones y supuestos

Existen una serie de circunstancias y restricciones que deben tenerse en consideración durante el desarrollo del proyecto:

- El número de miembros del equipo de desarrollo no variará durante el proyecto, tal y como se indicó en el apartado 2.5.3.
- Conexión a Internet de alta velocidad.
- Se realizará la formación adecuada en las herramientas de trabajo del proyecto si es necesario para el correcto desarrollo de éste.
- La planificación temporal del proyecto es de obligado cumplimiento por parte del desarrollador.

4.2.3. Entregas a realizar y contenido de las fases

Tal como se especifica en el apartado 2.7, las fases se dividen en las siguientes etapas, las cuales se encuentran incluidas en diversas iteraciones para marcar el orden y ritmo de ejecución de cada una. Además, se ha definido la fecha de entrega de cada una de estas etapas. Toda esta información se detalla en la tabla 4.1.

Fase e iteración		Etapa	Límite
Inicial	I1	Documentación y preparación del entorno de trabajo	27/01/2012
	I1	Especificación inicial	10/02/2012
Elaboración	E1	Planificación de la fase de elaboración - E1	14/02/2012
	E1	Especificación detallada de los casos de uso	13/03/2012
	E2	Planificación de la fase de elaboración - E2	15/03/2012
	E2	Diseño de la arquitectura del sistema	22/03/2012
	E2	Diseño de la arquitectura de la información	29/03/2012
	E2	Revisión de los casos de uso de la iteración E1	10/04/2012
	E2	Especificación definitiva de los casos de uso	09/05/2012
	E2	Primera aproximación del algoritmo de repetición	16/05/2012
Construcción	C1	Planificación de la fase de construcción - C1	17/05/2012
	C1	Implementación de la arquitectura `REST`	08/06/2012
	C1	Gestión de progreso	22/06/2012
	C1	Gestión de interfaz	06/07/2012
	C1	Etapa de pruebas intermedia	09/07/2012
	C1	Documentación de la implementación y pruebas	10/07/2012
	C2	Planificación de la fase de construcción - C2	11/07/2012
	C2	Revisión de los módulos implementados en C1	12/07/2012
	C2	Gestión de problemas	19/07/2012
	C2	Gestión de navegación	09/08/2012
	C2	Gestión de usuarios	17/08/2012
	C2	Etapa de pruebas	20/08/2012
	C2	Documentación de la implementación y pruebas	21/08/2012
	C3	Planificación de la fase de construcción - C3	22/08/2012
	C3	Revisión de los módulos implementados en C2	23/08/2012
	C3	Etapa de pruebas	24/08/2012
	C3	Documentación de las pruebas	27/08/2012
	C4	Planificación de la fase de construcción - C4	28/08/2012
	C4	Gestión de cursos	05/09/2012
	C4	Etapa de pruebas	10/09/2012
	C4	Definición del prototipo	13/09/2012
	C4	Documentación de la implementación y pruebas	14/09/2012
	C4	Preparación de la fase de transición	17/09/2012
Transición	T1	Planificación de la fase de transición	18/09/2012
	T1	Etapa de pruebas externas	25/09/2012
	T1	Detección de cambios	27/09/2012
	T1	Aplicar cambios	04/10/2012
	T1	Etapa final de pruebas	16/10/2012
	T1	Definición de la versión final del producto	17/10/2012
	T1	Documentación del sistema	24/10/2012
	T1	Documentación de ayuda al usuario	26/10/2012
	T1	Realización de la memoria del proyecto	19/11/2012
	T1	Revisar la coherencia "proyecto-documentación"	21/11/2012

Cuadro 4.1: Calendario del proyecto detallado: fases y etapas principales.

4.3. Organización del proyecto

4.3.1. Estructura organizativa

Tal y como se indicó en el apartado 2.5.3, hay una única persona desarrollando este proyecto, por lo que sus principales responsabilidades serán la de jefe de proyecto (siempre bajo la supervisión del director del proyecto final de carrera), la de analista y la de arquitecto.

4.3.2. Contactos externos

El desarrollador estará en contacto con posibles usuarios de la aplicación, con el director y con el ponente del proyecto para cualquier problema o duda que pueda surgir.

4.3.3. Roles y responsabilidades

Los roles y las responsabilidades asociadas recaen en una única persona, cuyas disciplinas `RUP` y habilidades serán adquiridas de forma conjunta para el desarrollo de este proyecto: *Project Management*, *Analysis & Design*, *Requirements*, *Environment*, *Business Modeling* y *Test*.

Las principales responsabilidades de cada rol son las siguientes:

- Jefe de proyecto: Búsqueda de recursos, asignación de prioridades, coordinación con los empleados y los clientes, a la vez que lidera el equipo de desarrollo hacia sus objetivos. Sus responsabilidades son controlar la calidad y la coherencia de los documentos producidos por el resto de roles del equipo de desarrollo.
- Analista: Responsable del modelo de casos de uso y del modelo de análisis.
- Arquitecto: Responsable de la arquitectura del sistema y diseño de los casos de uso más significativos.

4.4. Proceso de dirección

4.4.1. Introducción a la metodología empleada: identificación de pautas críticas

Es frecuente que exista ralentización de la planificación si el desarrollador se enfrenta por primera vez a una metodología nueva en la que tiene poca experiencia. A causa de esto, se ha planificado cada iteración de manera que existe un cierto margen para recibir la formación adecuada y poner en práctica dicha metodología (tal y como se expone en el apartado 2.5.3).

Una situación también muy frecuente es descubrir que la documentación es extensa y está muy dividida en diversos pasos según cada iteración, lo que lleva a concluir, erróneamente, que `RUP` sólo puede aplicarse a grandes proyectos. Estos hechos han llevado a elaborar una lista de las diez cosas imprescindibles en `RUP` [5], sus puntos esenciales, para así no caer en el error de dedicar demasiado tiempo a una tarea que no lo requiere o demasiado poco a aquella que sí lo necesita. Cada uno de estos diez puntos se detalla en un subapartado dedicado.

4.4.1.1. Desarrollar una buena visión general del proyecto

Es muy importante tener una idea clara de las necesidades reales del cliente, y esta información se ha de desarrollar en este documento: esbozando los requisitos esenciales, analizando el problema, definiendo el sistema y gestionando los cambios a medida que se presentan. Pero por encima de todo, poder dar una visión global del proyecto y del objetivo que se pretende conseguir con su desarrollo.

Es la base, pues, de los requisitos técnicos detallados. Ha de dar respuesta a las siguientes preguntas:

- ¿Qué términos clave existen? Ver anexo C
- ¿Qué problema hay que resolver? Ver apartado 2.4
- ¿Quiénes son los *stakeholders*? ¿Y los usuarios? ¿Qué necesitan? Ver apartados 2.5 y 2.5.4
- ¿Qué ofrece el producto a desarrollar? Ver apartado 2.3
- ¿Qué requisitos funcionales hay? Ver apartado 5.3
- ¿Qué requisitos no funcionales hay? Ver apartado 2.8.2
- ¿Qué restricciones de diseño existen? Ver apartado 2.5.3

4.4.1.2. Elaborar una buena planificación

Existe una máxima que afirma que "El producto será tan bueno como lo sea su planificación" [7]. En `RUP`, el documento *Software Development Plan* (principalmente redactado en el apartado 4.2) concentra mucha información dispersa preparada durante la fase inicial. Ésta debe mantenerse y actualizarse convenientemente durante todo el proyecto.

Este hecho esencial, en conjunción con el tercero, cuarto, quinto y octavo, capturan la esencia de la gestión de proyectos en `RUP`, que engloba tareas de concepción del proyecto, evaluación de riesgos y alcance, control, planificación y evaluación en cada iteración y fase.

4.4.1.3. Identificar y atenuar los riesgos

Es muy importante identificar y combatir los riesgos más importantes cuanto antes. Cada riesgo identificado ha de contar con su correspondiente plan de mitigación y de contingencia. El apartado 2.9 debe servir como herramienta de planificación y como base para las iteraciones. Además, debe ser necesaria su consulta a la hora de tomar decisiones o de realizar cambios en la planificación.

4.4.1.4. Controlar los errores e incongruencias

El análisis continuo de los datos es muy importante. Si se ha encontrado un obstáculo, es necesario establecer una fecha límite de actuación y comunicarla convenientemente a la persona responsable de su solución. Así mismo, los progresos y actualizaciones del estado del proyecto han de quedar documentados debidamente.

4.4.1.5. Consultar el *Business Use Case* frecuentemente

Los documentos *Business Use Case Model* y *Business Use Case Realization* de RUP proporcionan mucha información, desde un punto de vista empresarial, para determinar la viabilidad de un proyecto. También ayudan a justificar el proyecto y a establecer límites económicos (ver apartado 5.3).

4.4.1.6. Diseñar una arquitectura por componentes

RUP proporciona una manera metódica y sistemática de diseñar, desarrollar y validar esta arquitectura. Los pasos implicados en el flujo de trabajo de *Analysis & Design* incluyen definir una arquitectura de candidato, refinar la arquitectura, analizar el comportamiento y diseñar componentes del sistema.

Esto se concentra en el capítulo 3, donde se presentan múltiples visiones de la arquitectura del sistema. Cada uno de estos puntos de vista del sistema se refiere a un conjunto determinado de *stakeholders* en el proceso de desarrollo: los usuarios finales, los diseñadores, los gestores, etc. Por este motivo, este documento permite establecer unos puntos concretos de referencia respecto a la arquitectura del sistema, de forma que los miembros del equipo de desarrollo se puedan comunicar adecuadamente y tratar cualquier decisión significativa en este aspecto.

4.4.1.7. Desarrollar y probar incrementalmente

La esencia de los flujos de trabajo relacionados con la implementación y de las pruebas en RUP es incremental: construir y probar componentes de sistema durante el ciclo de vida de proyecto, produciendo entregas ejecutables al final de cada iteración después de la fase inicial. Al final de la fase de construcción, un prototipo arquitectónico estará disponible para su evaluación. En este proyecto en concreto estará disponible la aplicación *web* con todos los requisitos definidos anteriormente (apartado 5.3 para más detalles) en funcionamiento. Aún será necesario una etapa más de pruebas que permita detectar los cambios necesarios sobre ese prototipo y gestionarlos. Esto se producirá en la fase de transición, la cual permitirá acabar de perfilar la versión definitiva del producto.

4.4.1.8. Verificar y evaluar los resultados

Tal y como sugiere el nombre, el documento *Iteration Assessment* de RUP concentra los resultados de cada iteración, y especifica si se han cumplido los objetivos o no, incluyendo procesos a mejorar y fallos a evitar. Esto se corresponde con la metodología basada en iteraciones y con las etapas de pruebas, las cuales permiten detectar estos fallos y corregirlos.

4.4.1.9. Gestionar y controlar los cambios correctamente

Tan pronto como los usuarios obtengan el primer prototipo del producto (o incluso antes), éstos pedirán cambios. Es importante, pues, que los cambios propuestos se gestionen mediante un proceso consistente y se tenga en cuenta propagar los cambios a los documentos realizados.

4.4.1.10. Proporcionar documentación al usuario

Con RUP es muy importante el concepto de elaborar y entregar el producto iterativamente. Sería necesario, por lo tanto, entregar al usuario como mínimo una guía de usuario. En este caso, tal y como se apunta en el apartado 2.7, es necesario redactar documentación para usuarios, ayuda en línea y documentación técnica para administradores.

4.4.2. Planificación del proyecto

4.4.2.1. Planificación de las etapas

El desarrollo del proyecto estará determinado globalmente por fases, en las cuales se divide el trabajo en iteraciones, tal y como se ha comentado en el apartado 4.2.3. La tabla 4.2 muestra dichas fases en un formato compacto y resumido.

Fase	Número de iteraciones	Número de semanas	Fecha inicial	Fecha límite
Inicial	1	11	28/11/2011	10/02/2012
Elaboración	2	14	13/02/2012	16/05/2012
Construcción	4	19	17/05/2012	17/09/2012
Transición	1	10	18/09/2012	22/11/2012
Total	**8**	**54**	**28/11/2011**	**22/11/2012**

Cuadro 4.2: Calendario del proyecto resumido.

4.4.2.2. Planificación temporal del proyecto

Se ha realizado una planificación temporal del proyecto, y los detalles, se han plasmado en un diagrama de *Gantt* por cada fase del proyecto (ver anexo B).

4.4.2.3. Presupuesto del proyecto

Se ha calculado el coste asociado a cada perfil del equipo de desarrollo y a su rendimiento. El coste total del proyecto asciende, por lo tanto, a 75.000,00 €, tal y como se detalla en la tabla 4.3 (IVA incluido):

Este presupuesto se ha calculando en base a la consulta de los salarios actuales según la página *web* de InfoJobs[1]

[1]Más información en: http://salarios.infojobs.net/ .

Concepto	Porcentaje	Horas	Tarifa (€/hora)	Coste total
Definición de requisitos y prototipo	16,1 %	336	35,00	11.760,00
Análisis y diseño	30 %	624	40,00	24.960,00
Programación y pruebas unitarias	33,5 %	696	30,00	20.880,00
Pruebas de integración	5,4 %	112	30,00	3.360,00
Dirección y seguimiento proyecto	5 %	104	65,00	6.760,00
TOTAL Implementación	**90 %**	**1872**		**67.720,00**
Documentación de usuario y formación	7,3 %	152	30,00	4.560,00
Documentación de sistema	2,7 %	56	30,00	1.680,00
Imprevistos	0,8 %	16	65	1.040,00
TOTAL Proyecto	**100,8 %**	**2096**		**75.000,00**

Cuadro 4.3: Desglose de horas de desarrollo por fase y coste económico total.

Capítulo 5

Diseño del sistema

5.1. Introducción a la fase de elaboración

Una vez detallada la fase inicial en terminología RUP, que vendría a ser la fase de especificación del sistema, la siguiente etapa en el proceso de creación del producto *software* es la fase de elaboración. Tal y como se ha detallado en el apartado 2.7, esta fase consta de dos iteraciones, por lo que el texto aquí presentado es el resultado final después de pasar por todas ellas.

Es en esta etapa en la que se representa de manera abstracta el sistema que se implementará posteriormente, por lo que esta fase ha de asegurar que el producto final cumple con los requisitos enumerados previamente. Su objetivo es la definición del sistema con el nivel de detalle suficiente como para permitir su implementación, y permitirá determinar si la arquitectura es estable, si las pruebas efectuadas resuelven los riesgos y si el plan de proyecto es realista.

La fase de elaboración es la parte más crítica del proceso, ya que es al final de esta fase cuando se ha hecho la mayor parte de la ingeniería "dura" y donde se decide si se puede continuar con el proyecto de manera segura. Además, hay menos riesgos y se puede planificar el resto del proyecto con un menor grado de incertidumbre. Permitirá entrar en la siguiente fase, la de construcción, y centrarse en las pruebas y en la eficiencia, gracias a una planificación detallada y a una arquitectura madura y estable.

5.2. Arquitectura del sistema

Tal y como se describió en la fase anterior, la fase inicial, el sistema ha de cumplir una serie de requisitos funcionales y no funcionales (ver apartado 2.8 para más información sobre los requisitos). La arquitectura del sistema se basa en los capítulos 2 y 3.

A continuación se detalla el modelo de datos del sistema (imagen 5.1), indicando el identificador único de cada objeto, sus atributos propios y las referencias a otros objetos si es necesario.

5.3. Especificación detallada de los casos de uso

A continuación se definen los casos de uso del sistema. Se agrupan según los módulos en los cuales se implementarán y con los que serán relacionados. Éstas son las características básicas que serán descritas con detalle en cada uno:

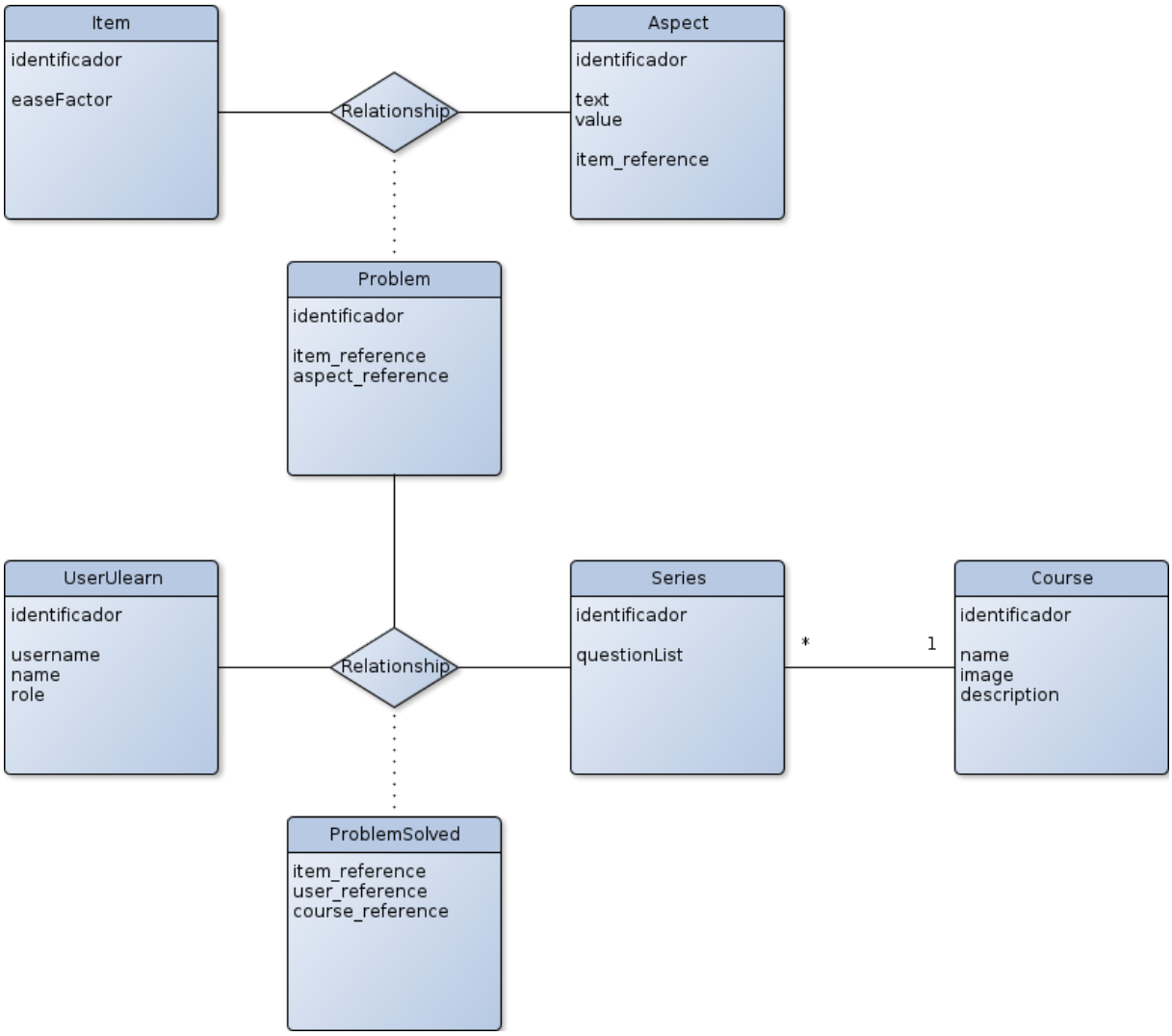


Figura 5.1: Modelo de datos (en la fase de elaboración).

- Actores
- Prioridad
- Descripción
- Pre-condición
- Post-condición
- Flujo normal
- Flujo alternativo

5.3.1. Gestión de interfaz (usuarios autenticados)

Para especificar los siguientes casos de uso se ha decidido tomar como caso base uno determinado a partir del cual se procede a especificar todos los demás, pues el funciona-

miento es exactamente el mismo: navegación a través de las diferentes partes de la página *web* y la información que se muestra al usuario en cada una de ellas.

5.3.1.1. consultarPerfil

Actores: Sistema, Usuario.

Prioridad: Media - Alta.

Descripción: El usuario consultará su perfil en la aplicación y el sistema se lo mostrará.

Pre-condición: El usuario está registrado y autenticado correctamente en el sistema.

Post-condición: El perfil se muestra al usuario.

Flujo normal:

1. El usuario selecciona la opción para consultar su perfil en la interfaz.
2. El sistema muestra el perfil al usuario.
3. Termina el caso de uso.

5.3.1.2. consultarCursoEnEstudio

Actores: Sistema, Usuario.

Prioridad: Media - Alta.

Descripción: El usuario consultará los cursos que actualmente está estudiando y el sistema se las mostrará.

Pre-condición: El usuario está registrado y autenticado correctamente en el sistema.

Post-condición: Los cursos en fase de estudio se muestran al usuario.

Flujo normal:

1. Igual que el caso de uso 5.3.1.1

Flujo alternativo: No hay cursos en fase de estudio (2)

3. El sistema muestra la lista de cursos vacía al usuario y se le explica el porqué. A continuación se le indica cómo poder comenzar a estudiar algún curso.
4. Termina el caso de uso.

5.3.2. Gestión de interfaz (usuarios no autenticados)

5.3.2.1. consultarCurso

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario consultará los cursos disponibles para el estudio y el sistema se las mostrará.

Pre-condición: No hay.

Post-condición: Los cursos disponibles se muestran al usuario.

Flujo normal:

1. El usuario selecciona la opción para consultar los cursos en la interfaz gráfica.
2. El sistema muestra la lista de cursos disponibles.
3. Termina el caso de uso.

Los siguientes tres casos de uso se han agrupado de forma que no se da una descripción detallada de los mismos puesto que el comportamiento descrito en el flujo normal del caso de uso 5.3.2.1 es el mismo en todos. La única diferencia es que el usuario se dirigirá a diferentes partes de la interfaz para consultar aquella información que desee en cada caso. Así pues, se listan a continuación los casos de uso para dar una idea de todo lo que el usuario puede consultar en la interfaz y se describen brevemente:

5.3.2.2. consultaVisitaGuiada

Descripción: El usuario se dirigirá a realizar una visita guiada por la página *web* y se le mostrarán diversas capturas de las partes más importantes de la misma.

5.3.2.3. consultaCaracterísticasULearn

Prioridad: Media - Baja.

Descripción: El usuario consultará información general acerca de porqué **ULearn** es una aplicación adecuada para aprender y memorizar conceptos de una manera diferente.

5.3.2.4. consultaVisualizarProgresoEjemplo

Prioridad: Media - Baja.

Descripción: El usuario consultará de qué manera podrá visualizar el progreso una vez que esté autenticado en el sistema.

5.3.2.5. consultaQuienesSomos

Descripción: El usuario consultará datos generales sobre **ULearn**.

5.3.2.6. contacto

Actores: Sistema, Usuario.

Prioridad: Media - Alta.

Descripción: El usuario contactará con los administradores del sistema.

Pre-condición: No hay.

Post-condición: El usuario habrá enviado satisfactoriamente un formulario de contacto al sistema.

Flujo normal:

1. El usuario selecciona la opción para poder rellenar los datos de contacto.
2. El sistema muestra el formulario de contacto al usuario.
3. El usuario rellena el formulario y lo envía.
4. Termina el caso de uso.

Flujo alternativo: No se ha enviado correctamente (3)

4. El usuario recibe un mensaje conforme no se ha podido enviar correctamente el formulario.
5. Termina el caso de uso.

Flujo alternativo: No se ha rellenado correctamente (3)

4. El usuario recibe un mensaje conforme no ha rellenado adecuadamente el formulario.
5. Termina el caso de uso.

5.3.3. Gestión de problemas

5.3.3.1. crearProblema

Actores: Sistema, Técnico.

Prioridad: Alta.

Descripción: El técnico creará un nuevo problema.

Pre-condición: El técnico tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.

Post-condición: Se crea un problema.

Flujo normal:

1. El técnico crea un problema nuevo dando al sistema todos los datos necesarios para ello.
2. Termina el caso de uso.

Flujo alternativo: No existe ningún curso (1)

2. El sistema indica que no hay ningún curso creado.
3. El técnico crea un curso (caso de uso 5.3.7.1)
4. Termina el caso de uso.

5.3.3.2. crearSerie

Actores: Sistema, Técnico.

Prioridad: Alta.

Descripción: El técnico creará una nueva serie de problemas para un curso.

Pre-condición:

- El técnico está autenticado y tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.
- Existe al menos un problema de un curso con el que poder crear una serie.

Post-condición: Se crea una nueva serie de problemas.

Flujo normal:

1. El sistema muestra al técnico los cursos disponibles.
2. El técnico selecciona el curso a la cual añadir los problemas.
3. El técnico crea la serie.
4. El sistema muestra al técnico los problemas disponibles.
5. El técnico selecciona un problema de los ya creados para dicho curso.
6. Se asocia el problema y la serie con el curso.
7. Se vuelve al punto 3 hasta que el técnico ha finalizado la serie.
8. Termina el caso de uso.

Flujo alternativo: El técnico decide crear una serie vacía (2)

3. El técnico crea una serie y no asocia ningún problema para la misma.
4. El sistema establece que la serie no sea visible para usuarios sin permisos.
5. Termina el caso de uso.

5.3.3.3. escogerCursoAEstudiar

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario escogerá un curso sobre la cual realizará problemas.

Pre-condición: El usuario está registrado y autenticado correctamente en el sistema.

Post-condición: Se informa al usuario que puede iniciar una serie de problemas a continuación.

Flujo normal:

1. El usuario escoge un curso de entre los disponibles.
2. El sistema le comunica el curso que ha escogido.
3. Termina el caso de uso.

5.3.3.4. resolverSerie

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario iniciará una serie de problemas pertenecientes a un curso determinado.

Pre-condición:

- El usuario está registrado y autenticado correctamente en el sistema.
- El usuario ha escogido con anterioridad un curso a estudiar.

Post-condición: La serie de problemas se inicia y el usuario puede comenzar a resolver problemas de la misma.

Flujo normal:

1. El usuario escoge una serie de problemas a practicar.
2. El sistema ejecuta el primer problema.
3. El usuario resuelve el problema.
4. El sistema guarda los datos estadísticos del problema resuelto.
5. Se vuelve al punto 2 hasta que finaliza la serie.
6. Termina el caso de uso.

Flujo alternativo: Se reanuda una serie no finalizada (1)

2. El sistema detecta que el usuario dejó una serie de problemas a medio hacer.
3. El sistema muestra el primer problema pendiente de la serie.
4. Se vuelve al punto 3.

Flujo alternativo: Se interrumpe la serie porque lo ha decidido el usuario (3)

4. El usuario no resuelve todos los problemas de la serie.
5. Se actualiza el progreso según los problemas que sí haya resuelto.
6. Se penaliza al usuario en su progreso por la interrupción de la serie.
7. Termina el caso de uso.

Flujo alternativo: Resolver problemas cuando el usuario no está autenticado. (1)

2. El usuario se desconecta del sistema.
3. El usuario escoge una serie de problemas a practicar.
4. El sistema le indica que no está autenticado o que no ha creado aún una cuenta y que debe hacerlo para empezar.
5. Termina el caso de uso.

5.3.4. Gestión de navegación

5.3.4.1. consultarProblema

Actores: Sistema, Técnico.

Prioridad: Media.

Descripción: El técnico desea consultar uno o varios problemas.

Pre-condición: El técnico está autenticado y tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.

Post-condición: El problema se muestra al técnico.

Flujo normal:

1. El técnico realiza la consulta del problema que desea.
2. El sistema muestra el problema consultado al técnico.
3. Termina el caso de uso.

Flujo alternativo: El problema no existe (2)

3. El problema consultado no existe y el sistema informa de ello al técnico.
4. Termina el caso de uso.

5.3.4.2. consultarSeries

Actores: Sistema, Técnico, Usuario.

Prioridad: Media.

Descripción: El técnico o el usuario desea consultar una o varias listas.

Pre-condición: El técnico o el usuario está autenticado.

Post-condición: La serie se muestra al técnico o al usuario.

Flujo normal:

1. El técnico o el usuario realiza la consulta de la serie que desea.
2. El sistema muestra la serie de problemas consultada al técnico o al usuario.
3. Termina el caso de uso.

5.3.4.3. consultarAspecto

Actores: Sistema, Técnico.

Prioridad: Media.

Descripción: El técnico desea consultar uno o varios aspectos de un problema.

Pre-condición: El técnico está autenticado y tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.

Post-condición: El aspecto se muestra al técnico.

Flujo normal:

1. El técnico realiza la consulta del aspecto que desea.
2. El sistema muestra el aspecto del problema consultado al técnico.
3. Termina el caso de uso.

Flujo alternativo: El aspecto no existe (2)

3. El aspecto consultado no existe y el sistema informa de ello al técnico.
4. Termina el caso de uso.

5.3.4.4. consultarRol

Actores: Sistema, Técnico.

Prioridad: Media - Baja.

Descripción: El técnico desea consultar uno o varios roles.

Pre-condición: El técnico está autenticado y tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.

Post-condición: El rol se muestra al técnico.

Flujo normal:

1. El técnico realiza la consulta del rol que desea.
2. El sistema muestra el rol consultado al técnico.
3. Termina el caso de uso.

Flujo alternativo:

3. El rol consultado no existe y el sistema informa de ello al técnico.
4. Termina el caso de uso.

5.3.4.5. consultarUsuario

Actores: Sistema, Técnico.

Prioridad: Media - Baja.

Descripción: El técnico desea consultar información acerca de uno o varios usuarios.

Pre-condición: El técnico está autenticado y tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.

Post-condición: Los datos del usuario se muestran al técnico.

Flujo normal:

1. El técnico realiza la consulta de los datos del usuario que desea.
2. El sistema muestra los datos del usuario consultado al técnico.
3. Termina el caso de uso.

Flujo alternativo: El usuario no existe (2)

3. El usuario consultado no existe y el sistema informa de ello al técnico.
4. Termina el caso de uso.

5.3.5. Gestión de usuarios

5.3.5.1. altaUsuario

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario accede por primera vez al sistema y desea registrarse.

Pre-condición: No hay.

Post-condición: El sistema recibe una solicitud correcta de alta de usuario.

Flujo normal:

1. El usuario introduce los datos requeridos por el sistema y acepta las condiciones de uso.
2. El sistema recibe la solicitud del usuario.
3. El sistema envía al usuario información al respecto del estado la misma.
4. Termina el caso de uso.

Flujo alternativo: Datos incorrectos (2)

3. El sistema indica al usuario que hay datos que no son válidos o correctos.
4. Volver al paso 1 del flujo principal.

Flujo alternativo: Error de envío del correo de confirmación (3)

4. El sistema notifica al técnico que ha habido un error en el envío de la confirmación de alta.
5. El técnico determina la acción a realizar para solucionar el problema.
6. Termina el caso de uso.

Flujo alternativo: Error de envío por dirección de correo inexistente (3)

4. El sistema notifica al técnico que ha habido un error en el envío de la confirmación de alta.
5. El sistema no confirma el alta de usuario.
6. Termina el caso de uso.

Flujo alternativo: El usuario ya existe (2)

3. El sistema indica al usuario que ya existe otro usuario con ese nombre y que debe escoger otro para seguir con el proceso de alta.
4. Termina el caso de uso.

5.3.5.2. confirmarAltaUsuario

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario ha de confirmar al sistema el alta iniciada en el caso de uso 5.3.5.1.

Pre-condición: El usuario ha iniciado el proceso de alta y ha enviado una solicitud, la cual ha sido recibida correctamente por el sistema.

Post-condición: El usuario se ha registrado correctamente en el sistema.

Flujo normal:

1. El sistema envía una confirmación de alta al correo electrónico proporcionado por el usuario.
2. El usuario confirma el alta siguiendo las instrucciones recibidas en el correo electrónico.
3. El sistema recibe la confirmación
4. El sistema registra correctamente al usuario.
5. Termina el caso de uso.

5.3.5.3. bajaUsuario

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario desea darse de baja del sistema.

Pre-condición: El usuario está registrado y autenticado en el sistema.

Post-condición: El usuario se ha dado de baja del sistema y se han eliminado todos sus datos.

Flujo normal:

1. El usuario indica al sistema su intención para darse de baja del servicio.
2. El sistema pide confirmación al usuario.
3. El usuario confirma la baja.
4. El sistema realiza la baja del usuario.
5. El sistema informa al usuario de que se ha procedido a darle de baja del sistema.
6. Termina el caso de uso.

5.3.5.4. bajaUsuarioTécnico

Actores: Sistema, Técnico.

Prioridad: Media - Baja.

Descripción: El técnico desea dar de baja a un usuario del sistema.

Pre-condición:

- El usuario está registrado en el sistema.
- El técnico tiene los permisos necesarios para realizar los cambios pertinentes en el sistema.

Post-condición: El usuario se ha dado de baja del sistema y se han eliminado todos sus datos.

Flujo normal:

1. El técnico indica al sistema su intención para dar de baja a un usuario determinado del servicio.
2. El sistema pide confirmación al técnico.
3. El técnico confirma la baja.
4. El sistema realiza la baja del usuario.
5. El sistema informa al usuario de que se ha procedido a darle de baja del sistema.
6. El sistema informa al técnico que se ha dado de baja al usuario satisfactoriamente del sistema.
7. Termina el caso de uso.

5.3.5.5. loginUsuario

Actores: Sistema, Usuario.

Prioridad: Alta.

Descripción: El usuario desea acceder al sistema.

Pre-condición: El usuario está registrado, pero no está autenticado en el sistema.

Post-condición: El usuario se ha autenticado correctamente en el sistema.

Flujo normal:

1. El usuario introduce su nombre de usuario y contraseña para poder acceder al sistema.
2. El sistema comprueba que el usuario es un usuario registrado y que los datos proporcionados son correctos.
3. El sistema da acceso al usuario al sistema.
4. Termina el caso de uso.

5.3.5.6. modificarPerfil

Actores: Sistema, Usuario.

Prioridad: Baja.

Descripción: El usuario modifica su perfil en la aplicación y el sistema muestra los cambios.

Pre-condición:

- El usuario está registrado y autenticado correctamente en el sistema.
- El usuario se ha dirigido previamente a consultar su perfil.

Post-condición: El perfil modificado se muestra al usuario.

Flujo normal:

1. El usuario selecciona la opción para modificar su perfil en la interfaz de consulta del perfil.
2. El usuario modifica sus datos.
3. El usuario confirma al sistema que desea guardar los cambios.
4. El sistema guarda los cambios realizados.
5. El perfil modificado se muestra al usuario.
6. Termina el caso de uso.

Flujo alternativo: El usuario no modifica su perfil (2)

3. El usuario decide no modificar su perfil y abandona el formulario.
4. Termina el caso de uso.

Flujo alternativo: Datos introducidos incorrectos (2)

3. El sistema indica al usuario que los datos introducidos son incorrectos.
4. El sistema permite al usuario modificar de nuevo los datos incorrectos.
5. Vuelve al punto 2 del flujo normal.

5.3.6. Gestión de progreso

5.3.6.1. consultarProgreso

Actores: Sistema, Usuario.

Prioridad: Media - Alta.

Descripción: El usuario desea consultar el progreso de su aprendizaje.

Pre-condición: El usuario está registrado y autenticado correctamente en el sistema.

Post-condición: Se muestra el progreso al usuario.

Flujo normal:

1. El usuario selecciona la opción para consultar el progreso en la interfaz del sistema.
2. El sistema muestra al usuario el progreso general del usuario en cada uno de los cursos que estudia.
3. Termina el caso de uso.

Flujo alternativo: Más detalle (2)

3. El usuario desea conocer en detalle el progreso en un curso.
4. El sistema muestra al usuario con más detalle el progreso en el curso escogido.
5. Termina el caso de uso.

5.3.6.2. consultarResultadoSerie

Actores: Sistema, Usuario.

Prioridad: Media - Baja.

Descripción: El usuario consulta el resultado de la serie de problemas una vez completada y el sistema se la muestra.

Pre-condición: El usuario está registrado y autenticado correctamente en el sistema.

Post-condición: El resultado de la serie se muestra al usuario.

Flujo normal:

1. El usuario selecciona la opción de consulta del resultado de la serie de problemas completada.
2. El sistema muestra el resultado de la serie al usuario
3. El sistema proporciona la opción de poder ver el progreso total del usuario (caso de uso 5.3.6.1).
4. Termina el caso de uso.

5.3.6.3. marcarProblema

Actores: Sistema.

Prioridad: Alta.

Descripción: El sistema marca el problema para que sea repetido según el cálculo del algoritmo que los espacia en el tiempo.

Pre-condición: No hay.

Post-condición: El problema queda marcado para su posterior repetición, según si se ha acertado o fallado.

Flujo normal:

1. El problema no es resuelto en la primera oportunidad por un usuario.
2. El sistema marca el problema para que sea repetido en un futuro cercano.
3. Termina el caso de uso.

Flujo alternativo: El problema es resuelto en la primera oportunidad (1)

2. El sistema marca el problema para que sea repetido antes de que el usuario llegue al umbral del olvido del mismo.
3. Termina el caso de uso.

5.3.6.4. actualizarProgreso

Actores: Sistema.

Prioridad: Alta.

Descripción: El sistema actualiza el progreso del usuario cada vez que éste realice una serie de problemas.

Pre-condición: Hay problemas resueltos de los cuales no se ha calculado el progreso que representa para el usuario.

Post-condición: El progreso se ha actualizado según los resultados del usuario.

Flujo normal:

1. El sistema consulta los resultados recogidos en el caso de uso 5.3.3.4.
2. El sistema actualiza el progreso global y por cursos del usuario que ha realizado la serie.
3. Termina el caso de uso.

Flujo alternativo: La serie no está terminada (1)

2. El usuario interrumpe la serie y no la finaliza.
3. El cálculo del progreso se hace tan sólo de aquellos problemas finalizados.
4. Se penaliza al usuario en su progreso por la interrupción de la serie.
5. En el momento de retomar el estudio de la serie, ésta seguirá por el problema por el cual se interrumpió.
6. Termina el caso de uso.

Flujo alternativo: La serie es interrumpida por motivos ajenos al usuario (1)

2. Se ha interrumpido la conexión con el sistema y por tanto se interrumpe la serie en fase de estudio.

3. En el momento de retomar el estudio de la serie, ésta seguirá por el problema por el cual se interrumpió.

4. Termina el caso de uso.

5.3.6.5. actualizarHabilidades

Actores: Sistema.

Prioridad: Baja.

Descripción: El sistema actualiza las habilidades del usuario cada vez que éste realice una serie de problemas.

Pre-condición: Hay problemas resueltos de los cuales no se ha calculado en qué grado cambian la habilidad que actualmente posee el usuario.

Post-condición: La habilidad se ha actualizado según los resultados del usuario.

Flujo normal:

1. El sistema consulta los resultados recogidos en el caso de uso 5.3.3.4.

2. El sistema calcula en qué nivel de habilidad ha mejorado el usuario al finalizar la serie.

3. La habilidad queda actualizada según corresponda.

4. Termina el caso de uso.

Flujo alternativo: La serie no está terminada (1)

2. El usuario interrumpe la serie y no la finaliza.

3. El cálculo de la habilidad se hace tan sólo a partir de aquellos problemas finalizados.

4. Termina el caso de uso.

Flujo alternativo: La serie es interrumpida por motivos ajenos al usuario (1)

2. Se ha interrumpido la conexión con el sistema y se interrumpe la serie.

3. Se informa al usuario del problema.

4. Termina el caso de uso.

5.3.6.6. actualizarLogros

Actores: Sistema.

Prioridad: Baja.

Descripción: El sistema actualiza los logros del usuario cada vez que éste realice una serie de problemas.

Pre-condición: Hay problemas resueltos de los cuales no se ha calculado en qué grado cambian los logros del usuario.

Post-condición: El logro se ha actualizado según los resultados del usuario.

Flujo normal:

1. El sistema consulta los resultados recogidos en el caso de uso 5.3.3.4.
2. El sistema calcula si el usuario ha superado algún logro al finalizar la serie y actualiza este dato.
3. Termina el caso de uso.

Flujo alternativo: La serie no está terminada (1)

2. El usuario interrumpe la serie y no la finaliza.
3. El cálculo del logro se hace tan sólo a partir de aquellos problemas finalizados.
4. Termina el caso de uso.

Flujo alternativo: La serie es interrumpida por motivos ajenos al usuario (1)

2. Se ha interrumpido la conexión con el sistema y se interrumpe la serie.
3. Se informa al usuario del problema.
4. Termina el caso de uso.

5.3.7. Gestión de cursos

5.3.7.1. crearCurso

Actores: Sistema, Técnico.

Prioridad: Alta.

Descripción: El técnico crea un nuevo curso.

Pre-condición: El técnico tiene los permisos necesarios para realizar los cambios pertinentes en el sistema

Post-condición: Se crea un curso.

Flujo normal:

1. El técnico decide qué nuevo curso crear.
2. El técnico realiza los cambios en el sistema necesarios.
3. Termina el caso de uso.

Flujo alternativo: El curso ya existe (2)

3. El sistema informa al técnico de que el curso que intenta crear ya existe.
4. Termina el caso de uso.

5.3.7.2. recepciónNuevoMaterial

Actores: Sistema, Técnico, Usuario.

Prioridad: Baja.

Descripción: Un usuario envía al sistema nuevo material que ha encontrado interesante para el estudio.

Pre-condición: El usuario está registrado y autenticado de forma correcta en el sistema.

Post-condición: Se acepta la petición de nuevo material y se marca para moderar.

Flujo normal:

1. El usuario envía al sistema material que considera adecuado para el estudio.
2. El sistema recibe la petición de aceptación de material y envía un aviso al técnico.
3. El material se almacena y se marca a la espera de ser moderado.
4. Termina el caso de uso.

5.3.7.3. moderarNuevoMaterial

Actores: Sistema, Técnico, Usuario.

Prioridad: Baja.

Descripción: El material enviado previamente por un usuario es moderado por el técnico para ver si se publica, se rechaza o se modifica.

Pre-condición:

- El técnico dispone de los permisos necesarios para realizar cambios en el sistema.
- Existe material marcado para moderar (caso de uso 5.3.7.2).

Post-condición: Se vuelve a marcar el material conforme si es apto para publicarse, si se modificará o si finalmente es rechazado.

Flujo normal:

1. El técnico inspecciona el material marcado como pendiente de moderación.
2. El técnico efectúa un juicio de valor respecto el material y toma un decisión.
3. Se vuelve a marcar el material con el nuevo estado que el técnico ha decidido darle.
4. Termina el caso de uso.

Flujo alternativo: El usuario ya no existe (1)

2. El técnico ha marcado el material como rechazado y no será publicado.
3. Termina el caso de uso.

Flujo alternativo: Material aceptado (3)

4. El técnico ha marcado el material como aceptable de ser publicado en el estado en el que lo envió el usuario.
5. El material se publica en el sistema para que otros usuarios puedan utilizarlo para el estudio.
6. Se avisa al usuario conforme se ha aceptado y publicado el material.
7. Termina el caso de uso.

Flujo alternativo: Material rechazado (3)

4. El técnico ha marcado el material como rechazado y no será publicado.
5. Se avisa al usuario conforme se ha rechazado el material.
6. Termina el caso de uso.

Flujo alternativo: Material modificado (3)

4. El técnico ha marcado el material como válido para ser publicado, pero con cambios respecto al material original que envió el usuario en un primer momento
5. El material se modifica según los criterios del técnico.
6. El material se publica en el sistema para que otros usuarios puedan utilizarlo para el estudio.
7. Se avisa al usuario conforme se ha aceptado el material con alguna modificación. También se le informa de que el material ha sido publicado.
8. Termina el caso de uso.

5.3.7.4. bajaMaterial

Actores: Sistema, Técnico, Usuario.

Prioridad: Baja.

Descripción: El técnico da de baja material aceptado en el sistema.

Pre-condición: El técnico dispone de los permisos necesarios para realizar cambios en el sistema.

Post-condición: Se acepta la petición de baja de material y se actualiza el sistema de forma pertinente.

Flujo normal:

1. El sistema muestra al técnico los materiales disponibles.
2. El técnico indica al sistema el material que desea eliminar.
3. El sistema recibe la petición para eliminar el material y pide confirmación al técnico.
4. El técnico confirma la baja del material.
5. El sistema elimina el material aportado.
6. El sistema actualiza las estadísticas y progresos de los usuarios que estuvieran estudiando dicho material o lo hayan completado.
7. En caso de que el material fuera aportado por un usuario existente en el sistema, se avisa a dicho usuario de que ese material ha sido eliminado.
8. Termina el caso de uso.

5.3.7.5. modificarCurso

Actores: Sistema, Técnico.

Prioridad: Baja.

Descripción: El técnico modifica un curso ya existente en el sistema.

Pre-condición: El técnico dispone de los permisos necesarios para realizar cambios en el sistema.

Post-condición: El curso se ha actualizado con los cambios realizados.

Flujo normal:

1. El sistema muestra al técnico los cursos disponibles en el sistema.
2. El técnico escoge un curso de los mostrados.
3. El técnico modifica los datos pertinentes.
4. El técnico confirma los cambios.
5. El sistema actualiza el curso con los cambios realizados por el técnico.
6. Termina el caso de uso.

5.4. Tipos de usuarios en el sistema

Existen diversos tipos de usuarios en el sistema para limitar los privilegios que puedan tener. En concreto hay tres tipos:

- Básico: Usuario que sólo tiene acceso a los cursos para poder estudiar su contenido. También puede consultar series de problemas e información parcial de problemas (para evitar mostrar la respuesta al usuario).
- Técnico: Usuario privilegiado que puede moderar nuevo material enviado por otros usuarios. Se ha decidido que las tareas de crear nuevos problemas, series y cursos se delegan en exclusiva al usuario "Gestor".
- Gestor: Usuario con acceso total y plenos privilegios.

5.5. Algoritmos ya existentes

Las aplicaciones de *Supermemo* y *Anki*[1] han sido los dos mayores referentes que ha tenido este proyecto a la hora de definir el algoritmo de repetición. Principalmente, este proyecto se ha basado en la versión 2 del algoritmo implementado por *Supermemo* o `SM2`. Partiendo de la base de que el concepto a estudiar ha de ser una unidad lo más pequeña posible, el algoritmo se basa en los siguientes puntos a la hora de planificar un ejercicio:

- El atributo `ease_factor` se inicializa a 2.5

[1] Algoritmo utilizado: `http://ankisrs.net/docs/dev/manual.html#_what_spaced_repetition_algorithm_does_anki_use` .

- Después de una pregunta, el usuario determina la dificultad de la misma valorándola del 0 al 5, donde el 0 representa un desconocimiento total de la pregunta y el 5 justo lo contrario.
- Después de cada repetición se modifica el atributo `ease_factor` teniendo en cuenta la valoración aportada por el usuario.
- Si la valoración recibida es menor a 3, el ítem se sitúa al inicio de la cola de aprendizaje, como si fuera a ser estudiado por primera vez.
- La función para calcular el número de días hasta la siguiente repetición del ítem, dada la repetición actual n, es la siguiente:

$$I(1) := 1$$
$$I(2) := 6$$
$$I(n) := I(n-1) * \texttt{ease\_factor}, \forall n > 2$$

El algoritmo implementado por *Anki* se diferencia de la aproximación realizada por el `SM2` en los siguientes aspectos:

- Una respuesta errónea hace que esa tarjeta se muestre indefinidamente en esa misma sesión de estudio hasta que se acierte, no necesariamente al final.
- El siguiente día se estudian 5 palabras nuevas más las que se respondieron erróneamente el día anterior.
- Las tarjetas que se están estudiando por primera vez no penalizan en exceso comparándolas con otras tarjetas ya estudiadas previamente. Es decir, diferencia entre el rendimiento del usuario en la fase de aprendizaje y la fase de retención.
- Tan sólo existen 4 valores para una tarjeta: 0 significa fallo, los otros 3 indican el diferente nivel de conocimiento que se tiene de la tarjeta en caso de acierto. El usuario valora cada problema después de haberlo resuelto.
- Si se ha respondido en un tiempo excesivo pero correctamente, se vuelve a repetir la palabra en la misma sesión.
- Se aplican prioridades a la hora de establecer un orden al mostrar las tarjetas.

Teniendo en cuenta estas dos variantes, el algoritmo que se ha utilizado en este proyecto combina características de ambos y adapta el `SM2` a variaciones ofrecidas por *Anki*. El hecho diferencial y no incluido en estos dos algoritmos es la **valoración automática de la respuesta del usuario**, por lo que se ha creado de cero. El algoritmo usado en **ULearn** está explicado convenientemente en el apartado 6.7.5.

Capítulo 6

Construcción del sistema

6.1. Introducción a la fase de construcción

La fase de elaboración ha permitido determinar que el sistema se puede construir y que la arquitectura es estable. El siguiente paso es la implementación del sistema diseñado. En consecuencia, al final de esta fase el sistema debe estar listo para su uso, por lo que la programación de código entra en escena y se convierte en la tarea principal a tener en cuenta a la hora de planificar.

El sistema permite ampliar fácilmente el número de materias a estudiar, así como de los problemas que forman las series de estudio de dichas materias. Esto es debido al diseño basado en la modularidad de las clases que representan el modelo de datos del sistema.

Se han definido cuatro iteraciones en esta fase, definiendo así una prioridad y un orden a las etapas y tareas a implementar. Al final de cada fase se obtendrá un sistema parcialmente acabado pero funcional.

6.2. Implementación general del sistema

Las principales etapas de esta fase y las tareas que las definen permiten efectuar la transición del diseño teórico a la implementación del producto. Se detallan a continuación:

- Describir cómo encaja la ejecución del sistema en el entorno de producción.
- Definir un modelo que permita implementar los casos de uso diseñados previamente.
- Mantener la coherencia y la integridad de la arquitectura, de manera que se aseguren:
  - La temprana reutilización de componentes y agrupación de funcionalidades.
  - La integración entre los elementos del diseño y los nuevos elementos que puedan ser requeridos.

6.2.1. Diferencias con la fase de elaboración

Se han implementado todos los casos de uso con prioridad igual o superior a "media - alta" excepto los siguientes:

"consultarAspecto" no se ha implementado porque el modelo de datos ha cambiado como se detalla en el apartado 6.7.2.

"confirmarAltaUsuario" no se ha implementado porque el servicio donde se aloja la aplicación ya ofrece la funcionalidad: los usuarios que se autentiquen han de disponer previamente de una cuenta de *Google*.

"bajaUsuario" no se ha implementado porque no ha tenido cabida en la planificación, dando prioridad a otros casos de uso.

Se han implementado los siguientes 17 casos de uso definidos en la fase de elaboración, a parte de los muchos nuevos que han ido surgiendo durante la implementación (por orden de aparición en el apartado 5.3): "consultarPerfil", "consultarCursoEnEstudio", "consultarCurso", "consultaQuienesSomos", "contacto", "crearProblema", "crearSerie", "escogerCursoAEstudiar", "resolverSerie", "consultarProblema", "consultarSeries", "altaUsuario", "loginUsuario", "consultarProgreso", "marcarProblema", "actualizarProgreso" y "crearCurso".

Una de las funciones imprescindibles para el funcionamiento del sistema es esta:

`solvingProblemsHandler` Este gestor se encarga principalmente de llevar a cabo el caso de uso "resolverSerie". Esta función se encarga de gestionar la interfaz en el momento en que el usuario está resolviendo ejercicios y de almacenar convenientemente los resultados.

Otras funciones estrechamente relacionadas con la anterior (sin caso de uso asociado directamente) e igualmente importantes son:

`determine_the_items_being_solved` Esta función retorna la información relacionada con los problemas que el usuario tiene pendiente de resolver en un momento determinado, si los hubiera.

`schedule` Este método es el encargado de planificar un problema ya resuelto para su próximo estudio.

`determine_next_series` Este método se encarga de definir el orden de las series que serán resueltas por el usuario. También tiene en cuenta en este orden las series que hayan podido ser interrumpidas antes de su finalización en la fase de estudio.

Los métodos llamados gestores o *handlers* son la parte esencial del código definido como "Controlador" en el patrón `MVC`. El código que corresponde a esta parte del patrón está implementado en el fichero `ulearn.py`. Estos *handlers* se encargan básicamente de gestionar las peticiones `GET` y `POST` que llegan desde el navegador y actualizan convenientemente la vista o el modelo. Previamente se ha hablado del gestor `solvingProblemsHandler` y a continuación se describe uno más; el funcionamiento del resto es fácilmente entendible al observar su nombre:

`AboutHandler` : este gestor tan sólo muestra la vista que incluye la información acerca del proyecto **ULearn** a grandes rasgos. Esta información es pública para cualquier usuario que visite la `URL http://ulearnpfc.appspot.com/about` .

Por otro lado, existe información en la aplicación que precisa de usuarios con privilegios para poder consultarla o modificarla. Así, por ejemplo:

admin_items_handler Este gestor muestra una vista con todos los problemas que hay creados en el sistema, sólo accesible a técnicos o a gestores. Permite navegar por las diferentes páginas de información, estableciendo claramente a través de la `URL` en qué estado se encuentra el usuario, manteniendo así uno de los principios definidos por `REST` (más información en el apartado 6.6.1).

admin_items_create_handler Este gestor, relacionado con el anterior, permite a un técnico o a un gestor la creación de problemas nuevos en el sistema.

Ha sido necesaria la creación de un caso de uso nuevo relacionado estrechamente con el caso de uso "consultarProblema" en el que un usuario sin privilegios en el sistema es capaz de consultar la información relacionada con un problema de forma parcial, de manera que pueda tener una idea inicial de que lo estudiará sin conocer la respuesta por adelantado.

Finalmente se presenta el modelo de datos implementado, representado en la figura 6.1.

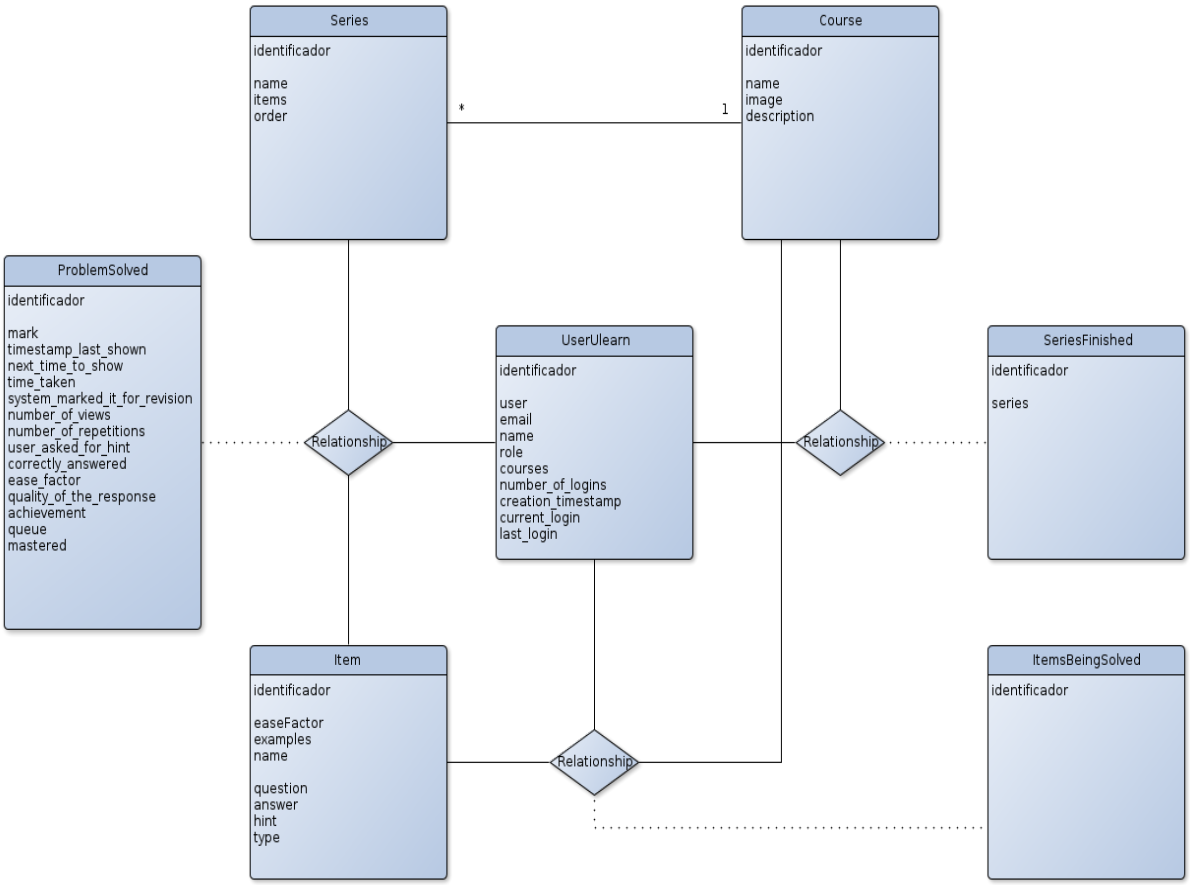


Figura 6.1: Modelo de datos.

6.3. El sistema en la nube

Google App Engine (también conocido por sus siglas `GAE`) es una plataforma que ofrece servicios en la nube del tipo *Platform as a Service*[1] (`PaaS`) y que permite desarrollar y alojar aplicaciones *web* en los centros de datos de *Google*.

El sistema se ha implementado utilizando este servicio de *Google* porque ofrece diversas ventajas:

- Es fácil comenzar un proyecto: el sistema es un proyecto nuevo que debe estar listo lo antes posible. Además, no es un proyecto muy grande ni tiene requisitos muy complicados de programar.
- El servicio es gratuito, lo cual permite el desarrollo sin la necesidad de realizar inversiones iniciales.
- Los límites son razonables: las limitaciones del servicio gratuito no son un problema para un proyecto de las características definidas en este documento.
- Escalabilidad automática: el servicio dispone de una función de escalabilidad automática independientemente del número de usuarios y de la cantidad de datos del sistema.
- Fiable y seguro: el sistema se ejecuta en entornos de ejecución controlados y se aplican las mismas políticas de seguridad, privacidad y protección de datos que a las demás aplicaciones de *Google*.

Para comenzar a desarrollar este sistema utilizando *Google App Engine* se necesitan estos requisitos:

- Conocer *Python* (aunque también están disponibles *Java* y *Go*[2]).
- Descargar el entorno de desarrollo de *Google App Engine* para *Python*.
- Instalar *Eclipse* y el *plugin PyDev*.
- Crear una cuenta en el servicio.

6.3.1. Límites y restricciones

Los límites del servicio para las cuentas gratuitas[3] son muchos pero suficientes para el desarrollo del sistema. Todos los límites son reiniciados cada 24 horas (excepto el de espacio consumido en disco) y se pueden consultar fácilmente a través de la página *web* de administración del sistema.

Los principales límites quedan resumidos en la tabla 6.1.

Existen otros límites, en muchos de los cuales se utiliza un control por periodos de tiempo de un minuto.

Además de las limitaciones por un servicio gratuito, existen otras restricciones. Estas restricciones existen por razones de diseño y no se pueden sortear pagando una mejora. Las más importantes son las siguientes:

[1] Más información en: `https://en.wikipedia.org/wiki/Platform_as_a_service` .
[2] Más información en: `https://en.wikipedia.org/wiki/Go_%28programming_language%29` .
[3] Más información en: `https://developers.google.com/appengine/docs/quotas` .

Concepto	Límite diario
Correos electrónicos	100
Ancho de banda	1 GB
Base de datos	1 GB
Operaciones sobre la base de datos	50.000

Cuadro 6.1: Resumen de los límites del servicio gratuito `GAE`.

- No existe un sistema de ficheros en el que almacenar datos, sólo se pueden almacenar en la base de datos.
- El servicio sólo responde a peticiones `HTTP`, `HTTPS`, `XMPP` y correos entrantes, no a otro tipo de peticiones.
- Se pueden usar módulos de fuentes externas al servicio, pero sólo si están escritos íntegramente en *Python*.
- La base de datos no acepta más de un modificador de búsqueda de desigualdad por clase de la base de datos en una consulta.
- El límite de tiempo de ejecución para tareas que no corran en segundo plano es de 60 segundos.

6.3.2. Ciclo de desarrollo con *Google App Engine*

Una manera de simplificar el desarrollo sobre este servicio de *Google* es el uso de algún entorno de desarrollo integrado de los soportados por el servicio. En esta ocasión se ha utilizado *webapp2*[4] por ser sencillo y fácil de aprender, además de uno de los que más ayuda tiene a través de los ejemplos de la documentación del servicio.

Una vez explicado esto, el primer paso es desarrollar la aplicación utilizando una herramienta de edición como *Eclipse* y el entorno de desarrollo de *Google App Engine* para *Python*.

Después se recomienda probar la aplicación localmente para comprobar que todo es correcto, a través del *script* ofrecido por el entorno de desarrollo de *Google App Engine* para *Python* de nombre `dev_appserver.py`.

Cuando todo sea correcto y se haya probado exhaustivamente, es el momento de subir la aplicación a los servidores de producción de *Google* para que los cambios y mejoras estén públicamente disponibles. De nuevo, existe un *script* ofrecido por el entorno de desarrollo, de nombre `appcfg.py`, que es necesario utilizar para esta tarea.

Por último, el ciclo vuelve a empezar mejorando o ampliando la aplicación a través de la herramienta de edición, probando los cambios y, de nuevo, publicando dichos cambios en los servidores públicos.

La figura 6.2 muestra de manera esquemática los pasos que se siguen en el proceso de desarrollo detallados anteriormente.

[4]Más información en `https://developers.google.com/appengine/docs/python/tools/webapp2` .

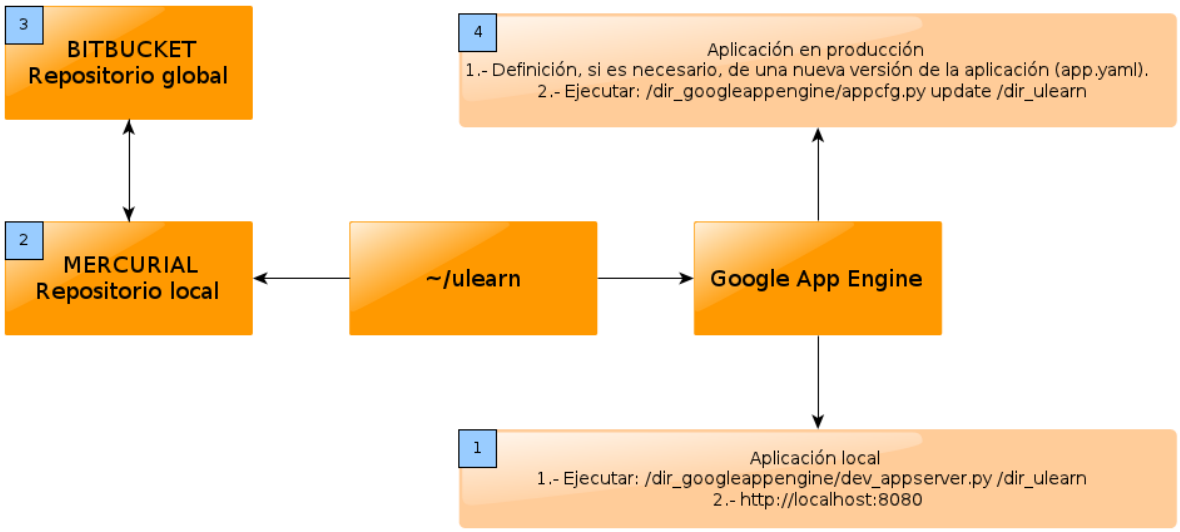


Figura 6.2: Esquema de los pasos de desarrollo.

6.3.3. La base de datos

La *App Engine Datastore* es la base de datos ofrecida por *Google App Engine*. El servicio ofrece una base de datos por aplicación y en la actualidad es del tipo *High Replication Datastore* (`HRD`, abreviadamente).

Se trata de una base de datos distribuida y totalmente tolerante a fallos preparada para gestionar grandes cantidades de datos, puesto que no sigue un esquema fijo para los datos que almacena. Se trata de una base de datos no relacional, por lo que es necesario un nuevo enfoque a la hora de diseñar aplicaciones que la usen de una manera óptima.

Además, la base de datos `HRD` ha estado funcionando ininterrumpidamente durante todo el año 2011[5]; y el porcentaje de horas asegurado funcionando es del 99'95 %[6].

6.3.4. Patrón `MVC` desde el punto de vista de *Google App Engine*

El uso del servicio de *Google* modifica este paradigma de programación (como se puede observar comparando con la figura 3.1), ya que el hecho de que todo el sistema esté en la nube implica que todo ha de pasar a través del controlador.

Tal y como se observa en la imagen 6.3, la esencia es la misma que la del patrón `MVC` tradicional pero modificada ligeramente.

6.4. El contacto con los administradores de la aplicación

Se ha implementado un formulario para que los usuarios que lo deseen, independientemente de si se han registrado en la aplicación o no, puedan enviar un comentario a los administradores del sistema (usuarios "técnico" o "gestor").

[5]Más información en: `http://googleappengine.blogspot.com.es/2012/01/happy-birthday-high-replication.html` .

[6]Más información en: `https://developers.google.com/appengine/sla` .

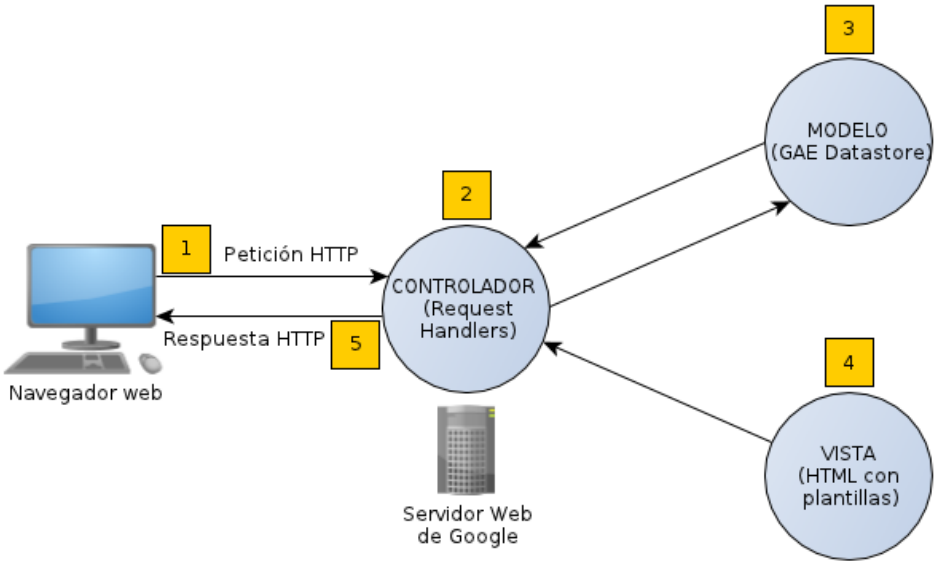


Figura 6.3: Paradigma `MVC` aplicado en el contexto de `GAE`.

Tan sólo es necesario introducir el nombre, una dirección de correo y el comentario que se considere necesario. Después será necesario resolver un fácil y divertido proceso de verificación para, a continuación, poder hacer clic en el botón que envía el comentario.

El proceso de verificación consiste en resolver un sencillo juego con el ratón en vez de tener que escribir palabras retorcidas y deformadas en una imagen, sin olvidar que a veces son difíciles o imposibles de escribir con el teclado. Este proceso es posible gracias al módulo explicado en el apartado 6.6.3.

6.5. Características del código del sistema

6.5.1. Documentación en el código con formato común

Se ha documentado cada función en profundidad junto a su cabecera siguiendo un único patrón, tal y como se detalla en el siguiente listado. De igual manera, se han documentado las instrucciones o grupos de instrucciones relevantes o importantes dentro de las funciones, así como todo aquello que sea necesario para una completa comprensión de las funciones.

- Descripción breve del propósito de la función. Si es necesaria más información, se hace a partir del primer punto y seguido.
- Línea en blanco.
- Lista de parámetros, ordenados y numerados, junto a una breve explicación separada por tabuladores. Si no hay parámetros, se indica claramente.
- Retorno de la función, explicación y tipo. Si no devuelve nada, se explicita claramente.

6.5.2. Registro de eventos

El servicio permite guardar un registro de eventos durante la ejecución del sistema, de manera que permite saber qué ha hecho en una determinada ejecución. Esta información es útil para los desarrolladores y se almacena durante 90 días con un tamaño máximo de 1 GB en los servidores de *Google*.

6.5.3. Constantes de sistema

Durante la implementación del sistema se ha considerado necesario definir un almacén único para las constantes necesarias en el código. Se implementa este almacén en el fichero `constants.py`, que incluye valores constantes para la ejecución de la aplicación, tales como dónde encontrar las plantillas del sistema o cómo acceder a los datos.

El uso de las constantes en las diferentes clases del sistema se realiza invocando la constante necesaria por su alcance completo, siempre que se haya cargado el almacén de constantes al inicio de la clase que lo necesite. En el siguiente ejemplo se declara la constante que almacena la ruta base de la que cuelgan todas las plantillas:

```
templates_dir = 'templates'
```

Y aquí se muestra cómo utilizar esta constante en cualquier punto del sistema:

```
import constants
 ...
constants.templates_dir
```

6.5.4. Diseño de las URLs válidas del sistema

Durante la implementación del sistema ha sido necesario decidir la estructura de la *web* y de sus URLs. Se ha utilizado el principio de que las URLs más sencillas son mejores[7], el cual, además, respeta unos de los principios básicos definidos en la arquitectura REST.

Otras ventajas que aporta esta aproximación es que son más fáciles de recordar y de escribir, si fuera necesario, por los usuarios, ayudando a la usabilidad y accesibilidad de la *web*. También pueden ayudar a mejorar la posición de la aplicación en los diversos buscadores *web* existentes[8].

Las siguientes direcciones, por ejemplo, son válidas para la aplicación y se han usado en el sistema definitivo:

`http://ulearnpfc.appspot.com/courses` → Muestra todos los cursos disponibles.

`http://ulearnpfc.appspot.com/admin/item/` → Muestra, a un usuario con privilegios, todos los problemas disponibles en el sistema.

En el apartado 6.2 se explica cómo se ha conseguido implementar esta funcionalidad.

Las siguientes direcciones también se emplean en el sistema, con la peculiaridad de que muestran los recursos del modelo de datos en HTML (páginas HTML):

[7] Más información en: `https://en.wikipedia.org/wiki/Clean_URL` .

[8] Más información en: `https://en.wikipedia.org/wiki/Search_engine_optimization` .

- Direcciones públicas:
  - /about: caso de uso “consultaQuienesSomos”.
  - /contact: caso de uso “contact”.
  - /help
  - /login: caso de uso “loginUsuario”.
  - /privacy: muestra la política de datos del sistema.
  - /tos: muestra los términos de uso del sistema.
  - /tour: caso de uso “consultaVisitaGuiada”.
  - /why_ulearn: no implementado.
- Direcciones para usuarios autenticados:
  - /admin
  - /desk
  - /desk/progress: caso de uso “consultarProgreso”.
  - /enroll/course_id: subscripción a un curso determinado.
  - /withdraw/course_id: desubscripción a un curso determinado.
  - /ulearn/course_id: sesiones de estudio de ejercicios.
  - /logout
- Direcciones descartadas durante la implementación del sistema:
  - /about_us: ahora es /about
  - /cancel
  - /cancel_ok
  - /create
  - /comunity
  - /my_desk: ahora es /desk
  - /exercise: ahora es /series
  - /help/first_time
  - /home
  - /home/exercise
  - /profile: ahora es /user
  - /progress_view
  - /settings: ahora es /user
  - /sign_up
  - /subject: ahora es /courses
  - /topic: ahora es /courses

6.6. Implementación del sistema

6.6.1. Definición de la arquitectura REST utilizada

REST, siglas de *REpresentational State Transfer*, es una manera de definir la arquitectura de un sistema, una guía para construir *web services*[9]. Uno de los motivos por el cual se ha utilizado este estilo arquitectónico es que REST se centra en usar la infraestructura **ya existente** en la *web* para el tráfico de los datos. De esta manera, se facilita la escalabilidad, se hace más sencilla la implementación y se construyen aplicaciones más reusables [8]. Otras fuentes que se han consultado para poder entender cómo definir la arquitectura según el estilo definido por REST han sido: [6] y [3]. Para entender qué es REST hace falta responder a las siguientes preguntas:

- **¿Qué son las URIs?** La nomenclatura más formal sería "¿Qué son los recursos?", relacionando la idea de que todo aquello que es identificado por una URI es un recurso. Con esto se ha llegado a la conclusión de que en el sistema implementado en este proyecto era necesario que cada recurso tuviera su propia URI. Y aquí es donde llega el momento de dividir el problema que queremos resolver hasta llegar a definir el tipo de recurso que queremos manipular, sabiendo que cada uno de esos recursos deberá tener su propia URI. Hay un ejemplo explicado en el apartado 6.5.4.

- **¿Cuál es el formato?** La nomenclatura más formal sería "¿Cuál es la representación?". Detrás de cada uno de estos recursos hay una manera de representarlos. En el caso de este proyecto, se representan en JSON.

- **¿Qué métodos son soportados por cada URI?** La nomenclatura más formal sería "¿Cómo se referencia una URI?". El acceso a un recurso identificado por una URI se puede hacer a través de las diferentes acciones que implementan los "verbos" HTTP. Estas acciones reciben el nombre abreviado de CRUD, que en inglés quiere decir: *Create, Retrieve, Update, Delete*. Hay que tener en cuenta que cualquier acción que no requiera una modificación del recurso se ha de hacer siempre con el método GET, es decir, para obtener un recurso siempre se ha de usar este verbo. Para la creación, POST; para la actualización o modificación de un recurso existente, PUT y para la eliminación de un recurso, DELETE. La correspondencia entre los verbos HTTP y las acciones CRUD está detallada en la tabla 6.2. De la misma forma, en la tabla que aparece en formato apaisado a continuación, se relacionan las acciones que se han definido con los verbos HTTP y con cada uno de los recursos que se han definido para este sistema en concreto.

- **¿Qué códigos de estado pueden retornar?** Una vez que se han decidido cómo serán las URIs, qué representación tendrán y qué acciones se podrán ejercer sobre los recursos, es necesario saber qué códigos de estado nos retornarán dichas acciones. De esta manera, a la hora de realizar pruebas sobre los recursos del sistema, se tendrá claro qué hay que esperar de uno u otro recurso. Esto también queda detallado en la tabla en la página siguiente.

Finalmente, un aspecto evidente que ha de tener todo recurso es que se pueda enlazar fácilmente con el siguiente recurso o con recursos distintos. Cuando en una página *web*

[9] En la tesis doctoral de Roy Fielding se describe REST como un principio arquitectónico de la *World Wide Web*: http://en.wikipedia.org/wiki/Roy_Fielding .

no podemos navegar fácilmente y se ha de hacer uso de las flechas de navegación del explorador o bien se ha de estar volviendo siempre a la página principal, existe una sensación de que se está perdiendo el tiempo, que la página puede que visualmente sea muy atractiva pero no facilita su uso y la consulta entre recursos a veces es tediosa. Formalmente hablando en términos de **REST**, a este aspecto se le denomina *Hypermedia as the engine of application state* (**HATEOAS**, abreviadamente) [13] [9]. En esencia, no se refiere tan sólo al hecho de enlazar recursos entre sí, si no el de permitir al cliente trasladar la aplicación de un estado a otro a través del conjunto de enlaces que se le facilitan, haciendo que el servidor no necesite almacenar ningún tipo de información sobre el estado de la aplicación (*stateless communication*).

Método HTTP	**Acción CRUD**	Descripción
POST	CREATE	Creación de un nuevo recurso.
GET	RETRIEVE	Obtener la representación de un recurso.
PUT	UPDATE	Actualizar un recurso.
DELETE	DELETE	Eliminar un recurso.

Cuadro 6.2: Correspondencia CRUD-HTTP.

En la siguiente lista se muestra el significado de la columna "Tipo" de la tabla en la que se muestra la relación entre los verbos HTTP y los recursos del sistema.

- U: Usuario sin privilegios
- A: Usuario con privilegios
- E: Fase de elaboración
- C: Fase de construcción

URL	TIPO	GET	PUT	POST	DELETE
/item	U	Acceso parcial	No implementado	No implementado	Sin acceso
	A	Mostrar recursos	No implementado	No implementado	Borrar recursos
	E	No implementado	No implementado	No implementado	No implementado
	C	Item::get_all ()	No implementado	No implementado	No implementado
/item/id	U	Acceso parcial	Sin acceso	Sin acceso	Sin acceso
	A	Mostrar recurso	Actualizar recurso	Crear recurso	Borrar recurso
	E	consultarProblema ()	No implementado	crearProblema ()	No implementado
	C	Item::get_item ()	Item::store ()	Item::store ()	No implementado
/series	U	Mostrar recursos	Sin acceso	Sin acceso	Sin acceso
	A	Mostrar recursos	No implementado	No implementado	Borrar recursos
	E	consultarSeries ()	No implementado	No implementado	No implementado
	C	Series::get_all ()	No implementado	No implementado	No implementado
/series/id	U	Acceso parcial	Sin acceso	Sin acceso	Sin acceso
	A	Mostrar recurso	Actualizar recurso	Crear recurso	Borrar recurso
	E	consultarSeries ()	No implementado	crearSerie ()	bajaMaterial ()
	C	Series::get_series ()	Series::store ()	Series::store ()	No implementado
/user	U	Sin acceso	Sin acceso	Sin acceso	Sin acceso
	A	Mostrar recursos	Actualizar recurso	Crear recurso	Borrar recursos
	E	consultarUsuario ()	No implementado	No implementado	No implementado
	C	UserUlearn::get_user ()	No implementado	No implementado	No implementado
/user/id	U	Mostrar recurso propio	Actualizar recurso propio	Crear recurso propio	Borrar recurso propio
	A	Mostrar recurso	No implementado	No implementado	No implementado
	E	consultarPerfil ()	modificarPerfil ()	altaUsuario () y confirmarAltaUsuario ()	bajaUsuario () y bajaUsuarioTécnico ()
	C	UserUlearn::get_user ()	UserUlearn::store ()	UserUlearn::store ()	No implementado
/courses	U	Mostrar recursos	No implementado	No implementado	Sin acceso
	A	Mostrar recursos	No implementado	No implementado	Borrar recursos
	E	consultarCurso ()	No implementado	No implementado	No implementado
	C	Course::get_all ()	No implementado	No implementado	No implementado
/courses/id	U	Mostrar recurso	Sin acceso	Sin acceso	Sin acceso
	A	Mostrar recurso	Actualizar recurso	Crear recurso	Borrar recurso
	E	consultarCurso ()	modificarCurso ()	crearCurso ()	No implementado
	C	Course::get_course ()	Course::store ()	Course::store ()	No implementado

6.6.2. Dotando al sistema de dinamismo

Lejos quedan los primeros años de la *web* en que todo el contenido era estático, a excepción de algunas imágenes o música. Los sitios *web* actuales son dinámicos y ricos en contenido, efectos y transiciones, para que la experiencia de usuario sea muy buena y vuelvan pronto o, en el peor de los casos, que no se vayan enseguida y se queden en él por un tiempo.

La interfaz gráfica del sistema implementado es una página *web* y, por lo tanto, ha de seguir los principios indicados en el párrafo anterior para atraer y mantener usuarios activos.

La mayor parte del dinamismo necesario para este sistema recae en la resolución de ejercicios, ya que ha de ser un proceso rápido y eficiente para que el usuario mantenga el interés en la aplicación y estudiar no sea tedioso o difícil. Estos objetivos se han conseguido a través del uso del lenguaje *Javascript*.

Este lenguaje es el único lenguaje común a todos los navegadores y el que mejor implementado está, por lo que es la mejor opción si se quiere acceder a la mayor cantidad posible de navegadores *web* y, por tanto, de usuarios que no encuentren problemas navegando por la aplicación.

El uso típico de *Javascript* es el de escribir funciones en la cabecera de las páginas `HTML` para interactuar con su estructura a través del *Document Object Model*[10]. En el sistema se utiliza en situaciones como las siguientes:

- Abrir una ventana emergente para las sesiones de estudio.
- Mostrar la ayuda de un problema si el usuario lo pide; en caso contrario, está oculta.
- Calcular el tiempo que tarda un usuario en resolver un problema.
- Validar los diversos formularios del sistema.
- Acceder a diversas funcionalidades a través de atajos de teclado: mostrar la ayuda (tecla ?), resolver un problema (tecla ↵).
- Mostrar al usuario si ha respondido bien o no un problema.
- Mostrar una barra de progreso que indica el tiempo restante. Notificar al usuario cuando el tiempo se ha acabado.

Javascript se ejecuta localmente en el navegador del usuario, no en el servidor del sistema, por lo que el sistema responde de una manera rápida a los eventos, proporcionando una buena experiencia de usuario.

La página *web* del sistema muestra un aviso si el navegador empleado para acceder a ella no soporta *JavaScript* e indica dónde encontrar ayuda para activarlo correctamente (`http://enable-javascript.com/`), ya que es una parte imprescindible para las sesiones de estudio de los usuarios:

```
ULearn makes use of JavaScript.
Please enable it in your browser's preferences.
If you don't know how to do this, go here.
```

[10]Más información en: `https://en.wikipedia.org/wiki/Document_Object_Model` .

6.6.3. Módulos externos utilizados

El sistema se ha implementado de cero y ha sido totalmente escrito en *Python* (versión 2.7.2, ya que las versiones superiores a la 3.0 no son soportadas por *Google*). También se han usado algunos módulos externos que han sido seleccionados para implementar algunas funcionalidades concretas y complejas. En concreto, los siguientes:

- *Google App Engine SDK*: El *Software Development Kit* de *Google* permite iniciar de una forma sencilla una aplicación y subirla a la nube en un tiempo muy corto. El `SDK` incluye diversos *frameworks* inicialmente tales como *Django* o *webapp2*, los cuales ofrecen las herramientas necesarias para combinar con las `API`s de *Google* y poder desarrollar fácilmente cualquier proyecto.

- *appengine-rest-server*: Convierte la aplicación en un servidor `REST` (`http://code.google.com/p/appengine-rest-server/`).

- *gae-sessions*: Proporciona un mecanismo sencillo y rápido para crear sesiones de usuario (`https://github.com/dound/gae-sessions`).

- *python-pagination*: Usado para crear y gestionar fácilmente la división de los resultados a mostrar en diferentes páginas cuando es necesario. Se dispone de una barra de navegación en la parte inferior de la página (`https://code.google.com/p/python-pagination/`).

- *python-dateutil*: Este pequeño módulo extiende la funcionalidad de la biblioteca que ya incorpora *Python* para las funciones de fecha y hora (`http://labix.org/python-dateutil`).

- *PlayThru*: Este módulo, disponible en *Python* y en muchos otros lenguajes, permite añadir un proceso de verificación humano a través de un sencillo juego, en vez de los incómodos *captchas*[11](`http://areyouahuman.com/`).

La aplicación también utiliza otros módulos pero, en este caso, para el código `HTML` y `CSS`:

- *Bootstrap*: Permite diseñar páginas web de manera rápida, ya que proporciona un estilo visual definido, mantenido y probado (`http://twitter.github.com/bootstrap/`).

- *Less CSS*: Amplia la funcionalidad de `CSS` y es necesario para usar *Bootstrap* (`http://lesscss.org/`).

- *Jinja2*: Herramienta para la construcción de los *templates* para *Python*. Ha sido necesario para realizar de "intermediario" entre el código del controlador y el de la vista. (`http://jinja.pocoo.org/`).

- *jQuery*: Proporciona elementos visuales complejos y/o animados. La librería *jQuery* (`http://jquery.com/`) es de las más utilizadas de *JavaScript*. Existen varios proyectos[12] creados a partir de *jQuery*, uno de los cuales es *jQuery UI*, utilizado en el proyecto. Añade efectos, interacciones y temas (entre otros) a la librería *jQuery* de base (`http://jqueryui.com/`).

[11] Más información en: `https://en.wikipedia.org/wiki/Captcha` .
[12] Más información en `http://jquery.org/` .

- *javascript-form-validation*: Comprueba que los datos introducidos en los formularios del sistema sean correctos, así como que todos los campos obligatorios hayan sido rellenados (`http://www.javascript-coder.com/html-form/javascript-form-validation.phtml`).

- *Google charts*: Las gráficas que muestran al usuario el progreso de los cursos que está estudiando se han realizado a través de la `API` de *Google* que permite de forma fácil implementar estas gráficas. (`https://google-developers.appspot.com/chart/`).

6.6.4. Persistencia de datos de la aplicación

Existen dos grandes tipos de datos que la aplicación ha de almacenar para un correcto funcionamiento:

- Los datos almacenados en la base de datos, como los problemas a resolver y los usuarios, han sido presentados brevemente en la figura 6.1 y discutidos en mayor profundidad en el apartado 6.6.8.

- La sesión de un usuario son todos aquellos datos que, desde que el usuario ha iniciado sesión en la aplicación hasta que la finaliza, garantizan la identificación del usuario en la aplicación y el acceso a sus estudios, evitando la identificación a cada paso que dé en la página web. Dentro de estos datos se encuentran el identificador único de cada usuario ligado a su cuenta de correo electrónico de *Gmail*, la lista de cursos suscritos o su nombre real.

A continuación se explica en mayor profundidad estos datos.

6.6.4.1. La base de datos

Los datos que se almacenan en la base de datos del servicio son accesibles a través de peticiones `HTTP` usando el servidor `REST` que se ha integrado en el sistema (ver apartado 6.6.3 para más información). La transferencia de información en ambos sentidos se realiza a través de un lenguaje que serializa estructuras complejas a través de la red: `JSON`.

El servidor `REST` permitía escoger entre dicho lenguaje y `XML`. La elección de `JSON` ha venido condicionada porque es capaz de transmitir la misma información usando una menor cantidad de caracteres.

El uso del módulo externo para el servidor `REST` ha tenido algunos daños colaterales en el rendimiento del sistema:

- Las lecturas y escrituras son ahora más lentas al añadir una nueva capa entre la aplicación y la base de datos.

- Tal y como está implementado el servicio, ha sido necesario duplicar la clave única de cada instancia debido a un problema de *dead-lock*: al guardar por primera vez una instancia, la clave única no se conoce hasta que la escritura finaliza correctamente y, a la vez, es necesario conocer la clave porque es la única manera de tener acceso a ella inmediatamente después a través del acceso por `REST`. Esto implica una segunda escritura de la instancia para almacenar la clave en otro atributo.

- Algunos campos han necesitado ser procesados antes de ser guardados para que el servidor de `REST` lo acepte:

  - Los atributos de tipo "fecha y hora" han de incorporar un carácter "T" entre la fecha y la hora en vez de un espacio en blanco.
  - Los atributos de tipo lista han de contener un único elemento de nombre "item" del que debe colgar la lista a guardar.

- Las comunicaciones con el servidor `REST` se realizan en `JSON`, por lo que hay que convertir los datos al enviarlos y convertirlos de nuevo en objetos de *Python* al recibirlos.

6.6.4.2. La sesión de usuario

Los datos de las sesiones, por el contrario, no se almacenan en la base de datos si no en *cookies*, una pequeña colección de datos que envían las páginas *web* y que almacenan y gestionan los navegadores. Esto proporciona, tal y como se ha explicado al principio de este apartado, una manera fácil de intercambiar datos en una interacción semipermanente, mejorando la experiencia de usuario y haciéndole la navegación más fácil.

Esto implica que el usuario ha de tener activadas las *cookies* en su navegador o, como mínimo, activarlas para el dominio de la aplicación y el de `google.com`.

No obstante, el uso de *cookies* en la aplicación impide cumplir plenamente el criterio de "*State Transfer*" de `REST` y, por lo tanto, nunca se podría afirmar que la aplicación es `RESTful`[10] (es decir, no cumple uno o más criterios `REST` explicados en → `https://en.wikipedia.org/wiki/Restful#Constraints`).

6.6.5. La problemática del diseño *web*

Los diseños previos de la interfaz gráfica del sistema han cambiado bastante en esta fase de construcción, ya que el uso de *Bootstrap* permite alcanzar un diseño más funcional y moderno de una manera sencilla.

Pese a las ventajas de su uso, es necesario utilizar exactamente los nombres de las clases `CSS` que *Bootstrap* ha definido y se ha intentado evitar este acoplamiento sin éxito a través de `LESS` (la extensión dinámica de `CSS`).

De haberse conseguido se evitarían posibles problemas futuros si *Bootstrap* cambiara nombres concretos de sus ficheros: un cambio de versión o de algún selector en concreto haría necesario un repaso exhaustivo del sistema actualizando todos y cada uno de los puntos en los que se insertó el código que ha sido modificado; mientras que si se hubiera creado un fichero propio que lo que hiciera fuera importar el código necesario de *Bootstrap* y adecuar los nombres de los selectores o clases a unos nombres propios, entonces únicamente se tendría que modificar dicho fichero realizando los cambios pertinentes según las novedades que se hayan producido en los ficheros de *Bootstrap*. De esta manera, el código `HTML` se desacoplaría mejor del propio de *Bootstrap*; tan sólo estaría acoplado al código propio desarrollado en el proyecto.

La conclusión es, por tanto, que se sigue adelante con el uso de *Bootstrap* a pesar del acoplamiento; cualquier cambio representará un análisis del código `HTML` del sistema en busca de posibles errores.

6.6.6. La implementación del código HTML

Todo el código HTML de la aplicación se ha construido usando la versión número 5, codificando las páginas en UTF-8 y validando satisfactoriamente su contenido por la W3C[13]. Esto garantiza que es código válido, bien estructurado y acorde a las especificaciones vigentes.

El código CSS que no pertenece a *Bootstrap* también ha sido validado satisfactoriamente.

6.6.7. La problemática de implementar sesiones de usuario

Google App Engine no ha implementado una gestión completa de sesiones en el entorno de desarrollo que ofrece. Esto significa que es necesario implementar ciertas funcionalidades a mano o bien buscar algún módulo externo que disponga de tales funcionalidades.

A continuación se comentan las opciones estudiadas y qué motivó que no fueran escogidas.

- *gae-init*: Es un *framework* completo y, por lo tanto, significa un cambio profundo en el proyecto (ya que no se trata de un módulo). Se ha intentado implantar en el proyecto (sin dejar de utilizar *webapp2*) para poder utilizar parte de sus funcionalidades sin éxito (`http://code.google.com/p/gae-init/`).

- *gae-utilities*: Se trata de un proyecto completamente descontinuado desde hace dos años. Tal y como indica su autor, la última versión no funciona para que nadie la use hasta que le pueda dedicar tiempo de nuevo (`http://gaeutilities.appspot.com/session`).

- *Django*: Se trata de otro *framework* completo y diferente a *webapp2*. Es un cambio profundo que implica una modificación importante del código imposible de asumir en la actualidad (`https://docs.djangoproject.com/en/1.4/topics/http/sessions/`).

- *webapp2*: Es el *framework* en el cual se ha implementado el sistema. Tal y como se ha indicado al principio de este punto, existe un mecanismo para la gestión de las sesiones pero aún es muy básico (`http://webapp-improved.appspot.com/api/webapp2_extras/sessions.html`).

La opción finalmente escogida es "*gae-sessions*", tal y como se ha indicado en el apartado 6.6.3.

Se trata de un módulo que implementa en un único fichero la gestión de las sesiones. El motivo para elegirlo es que, pese a que se trata de un proyecto descontinuado, funciona correctamente y cubre las necesidades del sistema [12].

6.6.8. Las clases que implementan el modelo de datos

Se han creado siete clases para implementar el modelo de datos de la aplicación, tal y como se ha presentado en la figura 6.1. A continuación se presenta el modelo de datos (figura 6.4) esta vez trasladándolo a las necesidades de la tecnología, normalizando el diagrama UML que se realizó en un primer momento.

[13]Más información en: `http://www.w3.org/` .

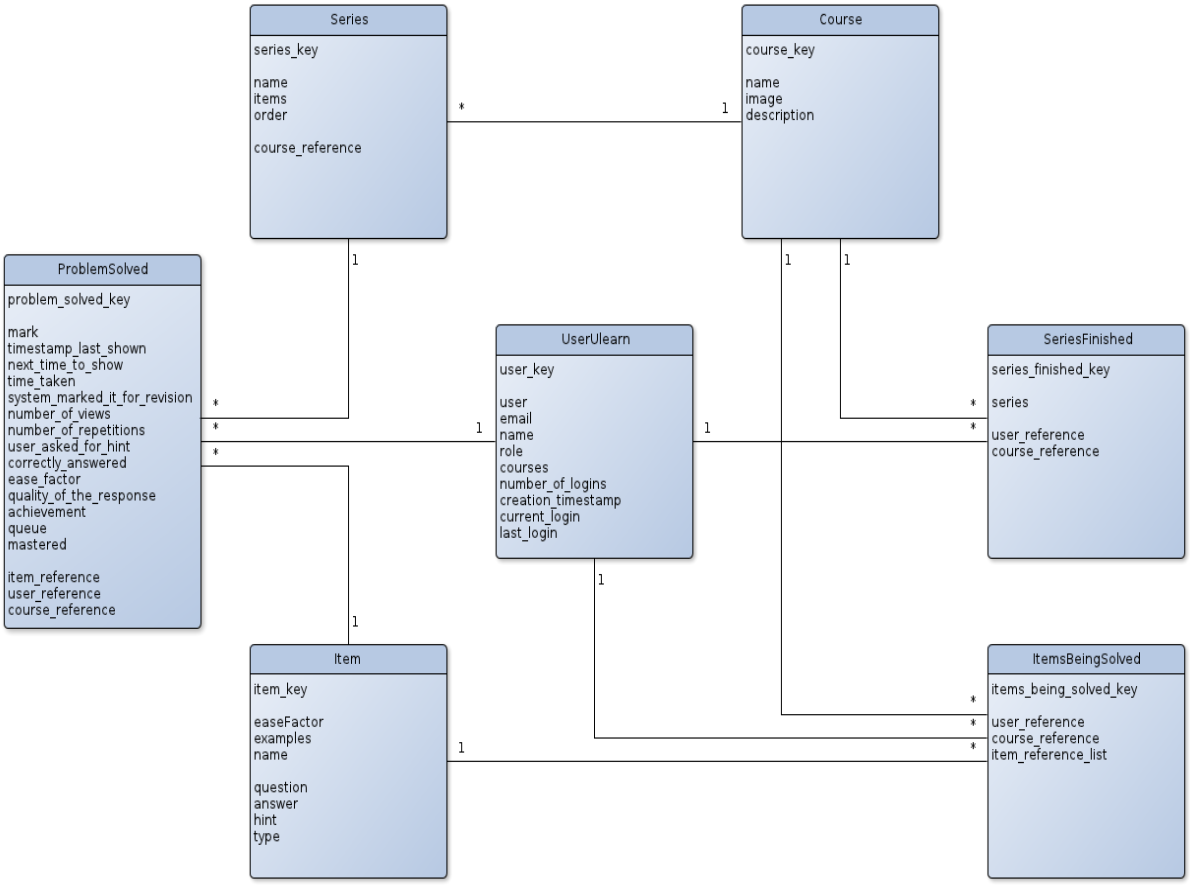


Figura 6.4: Modelo de datos normalizado.

Para conocer las funciones que se han implementado en cada clase del modelo, consultar el anexo E.

Seguidamente se detallan las funciones y sus atributos implementados a partir de este modelo de datos.

6.6.8.1. Clase "Item"

La clase `Item` representa un problema de la aplicación, una pregunta que el usuario debe responder durante el estudio. Es, por tanto, la unidad mínima de estudio para el usuario que se agrupa en series.

Los atributos de esta clase son los siguientes:

- `item_key`: Clave única que identifica cada instancia.
- `name`: Nombre que identifica a la instancia, en un formato fácil de entender por las personas.
- `easeFactor`: Factor de dificultad del problema. Se ha implementado como un entero entre 0 y 2 indicando dificultad baja, media y alta, respectivamente. El valor por omisión es "1" que equivale a una dificultad "media" (no confundir con el atributo `ease_factor` de la clase `ProblemSolved`). Se ha considerado que las palabras de 4

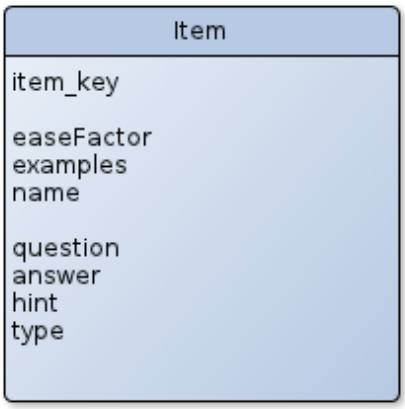


Figura 6.5: Clase `Item`.

letras o menos, son "fáciles" de responder; entre 5 y 8, tienen una dificultad "media" y más de 8 son "difíciles", en el caso de problemas de idiomas.

- `examples`: Conjunto de frases de ejemplo para el ítem. No es un campo obligatorio y pueden haber tantas como sean necesarias.
- `question`: Texto que se mostrará para formar la pregunta del problema.
- `answer`: Respuesta correcta para la pregunta.
- `hint`: Texto que se mostrará si el usuario solicita ayuda para resolver el problema.
  - En el caso de estudio de idiomas, se ha determinado que se mostrará la siguiente ayuda:
    - Respuesta de tres o menos letras: la primera letra de la palabra seguida de tantos caracteres de subrayado como la longitud de la palabra.
    - Respuesta compuesta por dos o más palabras: la primera y la última letra de cada palabra. Cada palabra contendrá tantos caracteres de subrayado como su longitud entre ambos extremos.
    - Problemas de dificultad alta: La primera, la última y la letra del medio de la palabra, rodeadas de tantos caracteres de subrayado como la longitud de los intervalos creados.
    - Resto de problemas: la primera y última letra de la palabra seguida de tantos caracteres de subrayado como su longitud.
  - En el caso de estudio de capitales del mundo, se ha determinado que se muestre la ayuda en los mismos términos que en el estudio de idiomas.
  - En el caso de finalizar letras de canciones, se ha determinado que se muestre entre una y tres de las primeras letras de la solución, hasta que la suma de sus caracteres sea inferior o igual a 9.
- `type`: Tipo de pregunta. Se ha implementado como una cadena cuyos únicos posibles valores son los seis que se han indicado en el apartado 6.7.1.

6.6.8.2. Clase "Series"

La clase `Series` representa un conjunto de problemas (de la clase `Item`). Es, por tanto, la unidad mínima para un usuario en una misma sesión de estudio en un momento determinado. Las series se agrupan en cursos.

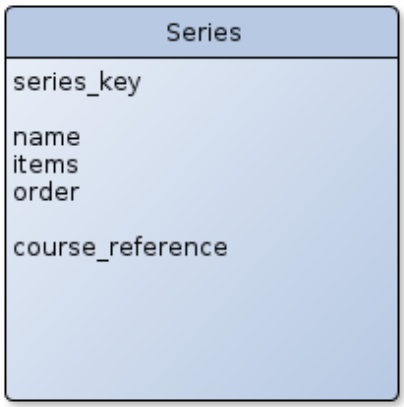


Figura 6.6: Clase `Series`.

Los atributos de esta clase son los siguientes:

- `series_key`: Clave única que identifica cada instancia.
- `name`: Nombre que identifica a la serie en un formato fácil de entender por las personas, que será mostrado al ver la información sobre el curso al que pertenece y al finalizar la serie.
- `items`: Conjunto de problemas (de la clase `Item`) que forman la serie.
- `order`: Número entero positivo que indica el orden de la serie dentro del curso, generalmente correlativos (se calcula automáticamente) comenzando en cero.
- `course_reference`: Clave única que identifica al curso al que pertenece la serie.

6.6.8.3. Clase "Course"

La clase `Course` representa a un conjunto de series (de la clase `Series`) para el estudio de los usuarios. Contiene, por tanto, series de problemas similares para que el usuario alcance un mayor conocimiento sobre un determinado tema.

Los atributos de esta clase son los siguientes:

- `course_key`: Clave única que identifica cada instancia.
- `name`: Nombre que identifica al curso y que será mostrado a los usuarios, en un formato fácil de entender.
- `image`: Imagen que ayuda a comprender rápidamente el contenido y los temas que trata un curso. En la versión actual del sistema se ha implementado como un enlace a una imagen externa a la aplicación, pero esto puede cambiar en el futuro.

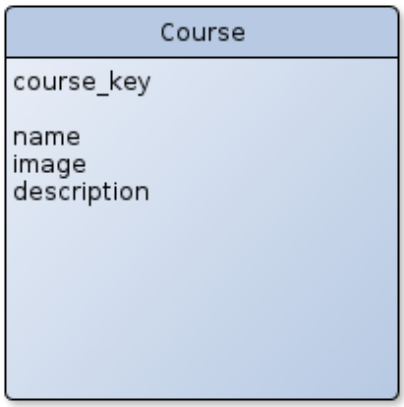


Figura 6.7: Clase `Course`.

- `description`: Descripción del contenido del curso, que será mostrado a los usuarios que accedan a la información detallada del mismo.

6.6.8.4. Clase "UserUlearn"

La clase `UserUlearn` identifica de manera inequívoca a los usuarios del sistema. Almacena información relacionada con el inicio de sesión, su rol dentro del sistema y qué cursos tiene suscritos en un momento determinado, entre otros datos. Parte de esta clase consiste en datos proporcionados por el servicio que es necesario almacenar.

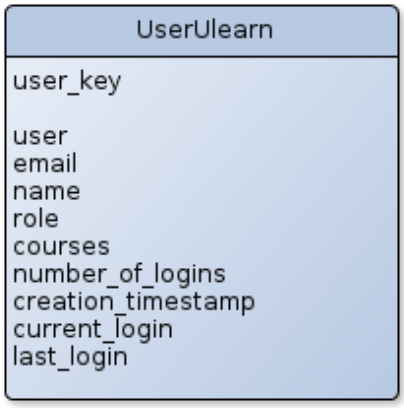


Figura 6.8: Clase `UserUlearn`.

Los atributos de esta clase son los siguientes:

- `user_key`: Clave única que identifica cada instancia dentro de la base de datos.
- `user`: Identificador único para cada usuario del sistema. Este atributo es proporcionado por *Google* y es el único que se garantiza que no cambia en una sesión de usuario.

- `email`: Dirección completa del correo electrónico del usuario, implementado como una cadena de texto. Este campo es proporcionado por el servicio.
- `name`: Nombre completo del usuario. En caso de que un usuario no lo haya definido en su cuenta de *Google*, su valor es el de la cuenta de correo sin el dominio ni el símbolo "@".
- `role`: Rol del usuario en el sistema. Se ha implementado como uno de estos posibles valores: "*user*", "*engineer*" o "*administrator*". Su función se ha definido en el apartado 2.5.
- `number_of_logins`: Número de veces que el usuario ha iniciado sesión de manera correcta.
- `creation_timestamp`: Fecha y hora del momento en que el usuario inició sesión por primera vez (ya que, tal y como se ha comentado en el apartado 6.2.1, en esta versión del sistema no se implementa el alta de usuario porque ya lo proporciona el servicio de *Google*).
- `current_login`: Fecha y hora de la última vez que el usuario inició sesión de manera correcta.
- `last_login`: Fecha y hora de la penúltima vez que el usuario inició sesión de manera correcta.
- `courses`: Conjunto de cursos a los que un usuario está suscrito en un determinado momento.

Esta clase está relacionada con las sesiones de usuario, explicadas en el apartado 6.6.4.2.

6.6.8.5. Clase "ItemsBeingSolved"

La clase `ItemsBeingSolved` representa el progreso de un usuario estudiando en un momento dado ya que, cuando un usuario inicia una nueva sesión de estudio, se almacenan los problemas que ha de resolver (ya sean nuevos o problemas que ha de volver a estudiar según la planificación del sistema).

Conforme el usuario resuelve correctamente problemas, éstos van siendo eliminados de la lista de problemas pendientes. Cuando ya no quedan problemas pendientes, los datos de ese usuario para ese curso se borran de esta clase.

Los atributos de esta clase son los siguientes:

- `items_being_solved_key`: Clave única que identifica cada instancia.
- `user_reference`: Clave única que identifica al usuario al que pertenece la instancia.
- `course_reference`: Clave única que identifica al curso al que pertenece la instancia del curso.
- `item_reference_list`: Conjunto de problemas decreciente en el tiempo a resolver por el usuario en el curso seleccionado.

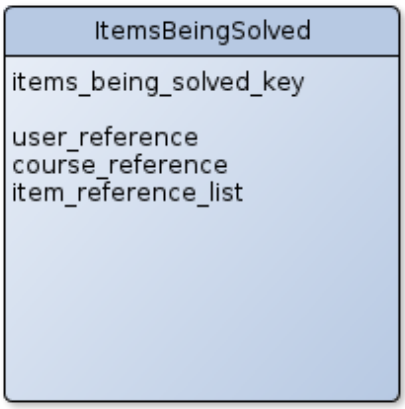


Figura 6.9: Clase `ItemsBeingSolved`.

6.6.8.6. Clase "ProblemSolved"

La clase `ProblemSolved` almacena una instancia de cada problema resuelto por un usuario de un determinado curso. En ella se almacenan datos sobre la calidad de su respuesta, el tiempo empleado y otros datos estadísticos, así como la fecha mínima en la que debe volver a estudiar el problema (ver apartado 6.7.5 para más información sobre la planificación).

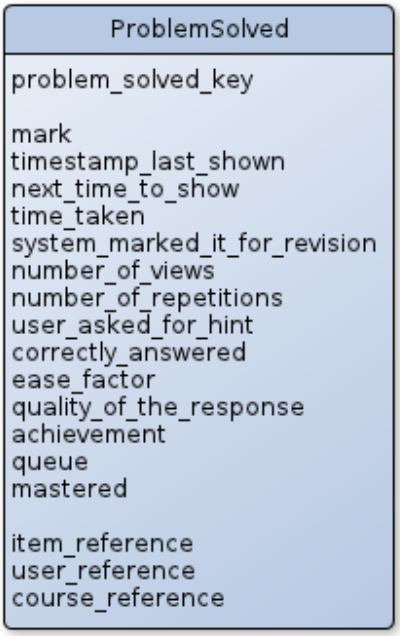


Figura 6.10: Clase `ProblemSolved`.

Los atributos de esta clase son los siguientes:

- `problem_solved_key`: Clave única que identifica cada instancia dentro de la base de datos.

- `mark`: Nota otorgada al usuario en el problema. Se ha implementado como un número decimal entre 1 y 10 (mínima y máxima nota, respectivamente). Su valor por omisión es 5,0.

- `timestamp_last_shown`: Fecha y hora en la que el usuario realizó el problema por última vez.

- `next_time_to_show`: Fecha y hora mínima en la que el usuario debe realizar el ejercicio de nuevo. Pese a que sólo es necesaria la fecha para este propósito, se guarda también la hora porque el servicio implementa internamente los atributos de fechas como un atributo de "fecha y hora" de todas formas.

- `time_taken`: Tiempo empleado por el usuario en la resolución del problema, medido en segundos. Se ha implementado como un entero positivo porque no es necesaria una mayor precisión.

- `system_marked_it_for_revision`: Indica si el sistema ha determinado que el usuario debe volver a estudiar el problema o no. Su valor se establece a "falso" si el usuario ha respondido correctamente y dentro del tiempo establecido; en cualquier otro supuesto se establece a "cierto". El valor por omisión es "falso".

- `number_of_views`: Número total de veces que el usuario ha estudiado el problema. Se ha implementado como un número entero y se inicia a cero.

- `number_of_repetitions`: Número de veces consecutivas que el usuario ha respondido correctamente el problema desde la última vez que lo falló. Se ha implementado como un número entero y se inicia a cero. Juega un papel clave en el proceso de determinar la fecha en la que el usuario ha de volver a estudiar un problema (función `determine_next_time_to_show()` de la clase `ProblemSolved`).

- `user_asked_for_hint`: Indica si el usuario ha solicitado la ayuda durante la resolución de un problema o no (valores "cierto" o "falso"). Su valor por omisión es "falso".

- `correctly_answered`: Indica si el usuario ha respondido correctamente al problema la última vez que lo estudió o no. Se ha implementado como un valor *booleano* establecido por omisión a "falso".

- `ease_factor`: Dificultad que el problema tiene para el usuario. Es un valor que cambia con cada nueva aparición del problema. Se ha implementado como un número decimal con un valor mínimo de 1,3 y su valor inicial es 2,5; en el apartado 6.7.5.1 se dan más detalles sobre este atributo.

- `quality_of_the_response`: Parámetro que indica cómo de buena ha sido la respuesta del usuario en el último estudio del problema. Se trata de un valor decimal entre 0 y 5 asignado según el baremo que se indica en el apartado 6.7.5.1; su valor por omisión es cero.

- `achievement`: Indica si el usuario ha resuelto el problema en una serie de supuestos de dificultad creciente. Actualmente sin un uso determinado en la versión actual del sistema. Su valor inicial es "falso".

- `queue`: Indica la cola de aprendizaje en la que se encuentra el problema para el usuario actual. Se ha implementado como un entero entre 1 y 5, siendo 1 la cola inicial y el valor por omisión. Hay más información sobre este sistema en el apartado 6.7.5.1.

- `mastered`: Parámetro que indica si el usuario conoce el ítem y lo recuerda perfectamente. No tiene un uso determinado en la versión actual del sistema. Su valor inicial es "falso".

- `item_reference`: Clave única que identifica al problema (de la clase `Item`) al que pertenece la instancia.

- `user_reference`: Clave única que identifica al usuario al que pertenece la instancia.

- `course_reference`: Clave única que identifica al curso al que pertenece la instancia.

6.6.8.7. Clase "SeriesFinished"

El propósito de la clase `SeriesFinished` es almacenar qué series ha acabado correctamente un usuario de un curso determinado, para que el sistema sepa cuáles ha de mostrar y cuáles no cuando el usuario empiece una nueva sesión de estudio.

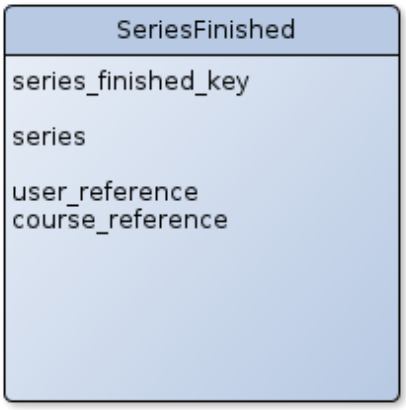


Figura 6.11: Clase `SeriesFinished`.

Los atributos de esta clase son los siguientes:

- `series_finished_key`: Clave única que identifica cada instancia.

- `series`: Conjunto de series que un usuario ha acabado del curso especificado por el atributo `course_reference`.

- `user_reference`: Clave única que identifica al usuario al que pertenece la instancia.

- `course_reference`: Clave única que identifica al curso al que pertenece la instancia del curso.

6.7. El estudio: implementación y algoritmo

6.7.1. Tipos de pregunta posibles

Durante la fase de construcción se han creado siete tipos de preguntas diferentes, dos más que en la fase inicial 2.1:

- **Audio**: La pregunta es un sonido que hay que escuchar y, a continuación, escribir lo que se ha escuchado como respuesta.
- **Image**: Se muestra una imagen y es necesario escribir como respuesta la palabra que la representa.
- **Capital**: En este caso la pregunta es el nombre de una capital de un país y es necesario responder de qué país es la capital.
- **Finish the lyric**: Se debe completar la frase inacabada de una letra de canción.
- **Phonetics**: Se presenta la pronunciación de una palabra según el Alfabeto Fonético Internacional[14] y es necesario responder de qué palabra se trata.
- **Translation**: La pregunta es una palabra en el idioma origen del usuario y hay que traducirla al idioma que se está estudiando.
- **Word**: Se presenta una palabra en el idioma que se está estudiando y es necesario indicar su traducción al idioma origen del usuario.

Cada tipo de pregunta se ha implementado en una plantilla para que los posibles cambios visuales no afecten al código y sean sencillos de identificar y modificar.

Debido a que esta primera versión del sistema ha sido creada principalmente para estudiar idiomas, todos los tipos de pregunta excepto el tipo *Capital* se han creado para este menester. La creación del tipo *Capital* se ha creado para cursos de geografía enfocados al aprendizaje de capitales de países.

6.7.2. Implementación inicial del algoritmo

Inicialmente se partió de un modelo de datos concreto para el desarrollo del proyecto que consistía en las siguientes clases: usuarios, ítems, listas y aspectos. En estas clases, se definió un ítem como parte del contenido que se desea aprender, ya que esta primera versión del sistema se creó principalmente para mejorar el estudio de otros idiomas. De cada ítem se desprendían unas cualidades denominadas aspectos, que eran la unidad mínima de aprendizaje y la base sobre la cual se elaborarían los problemas que más tarde resolvería el usuario. Estas cualidades se referían a la traducción del ítem en ambos sentidos, su pronunciación, su relación con un símbolo o su escucha. Además, las listas servían para organizar los ítems en diferentes categorías e iban a ser un pilar fundamental para calcular el progreso del usuario en la resolución de problemas.

Seguidamente se definieron dichos problemas como una serie de preguntas relacionadas con una misma palabra. Durante un breve espacio de tiempo, el sistema se creó sobre la base de que era necesario realizar todas las preguntas sobre un mismo problema en la

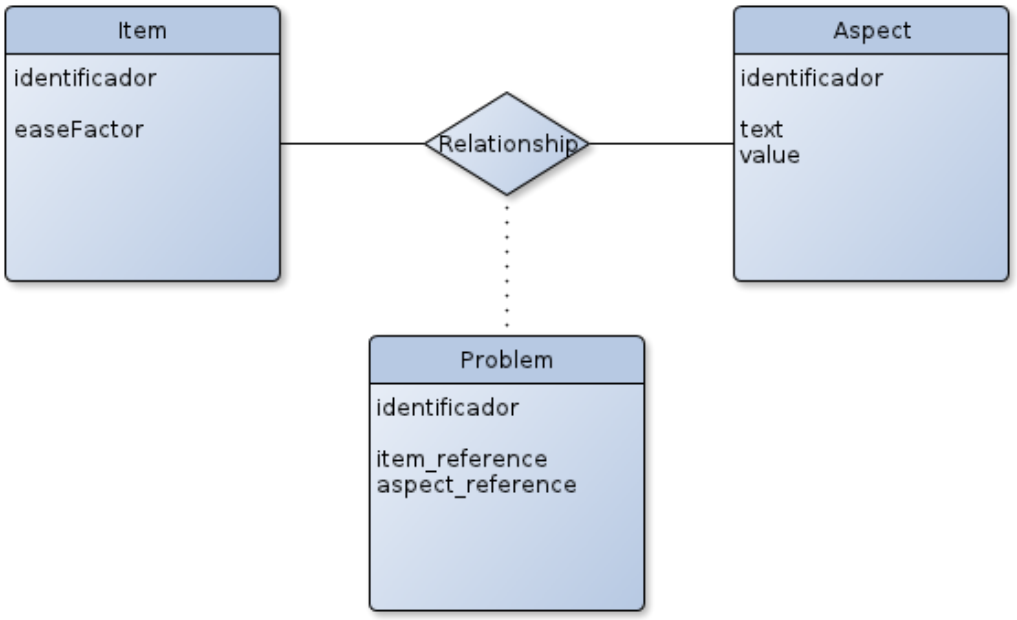


Figura 6.12: Modelo de datos `Item-Aspect-Problem`

misma sesión de estudio; ello derivó en el siguiente esquema del modelo de datos (vista parcial, ver la imagen 5.1 para la vista completa):

Este modelo de datos dividía un problema en dos de las clases comentadas previamente: la clase `Item`, la cual almacenaba las características comunes a todas las preguntas (en este caso, el factor de dificultad de la palabra original), y la clase `Aspect`, que guardaba todas las preguntas relativas a la palabra original. Para ello era necesario usar los campos `text` y `value`: el primero almacenaba el tipo de pregunta y el segundo la respuesta correcta.

En la tercera clase de la imagen 6.12, de nombre `Problem`, se relacionaban las preguntas de la clase `Aspect` con la palabra original de la clase `Item`.

Por ejemplo, el siguiente era un ejercicio válido para la palabra inglesa *house* (los valores del atributo "identificador" son valores inventados para este ejemplo):

Clase `Item`:

- identificador: house
  - `easeFactor`: 0

Clase `Aspect`:

- identificador: Aspect1
  - `text`: Audio
  - `value`: (Ruta al fichero de audio)
- identificador: Aspect2
  - `text`: Image

[14]Más información en: `https://en.wikipedia.org/wiki/IPA` .

    - `value`: (Ruta al fichero de imagen)
- identificador: Aspect3
    - `text`: Phonetics
    - `value`: /haʊs/
- identificador: Aspect4
    - `text`: Translation
    - `value`: house
- identificador: Aspect5
    - `text`: Word
    - `value`: casa

Clase `Problem`:

- identificador: Problem1
    - `item_reference`: house
    - `aspect_reference`: Aspect1
- identificador: Problem2
    - `item_reference`: house
    - `aspect_reference`: Aspect2
- identificador: Problem3
    - `item_reference`: house
    - `aspect_reference`: Aspect3
- identificador: Problem4
    - `item_reference`: house
    - `aspect_reference`: Aspect4
- identificador: Problem5
    - `item_reference`: house
    - `aspect_reference`: Aspect5

Esta aproximación tenía una serie de problemas que no tenían una fácil solución:

- Se guardan muchos datos para almacenar diversas preguntas sobre una palabra (en el ejemplo, se crean 11 instancias para tan sólo cinco preguntas reales).
- Acceder a la información completa sobre una palabra y todas sus preguntas es muy costoso en tiempo y en llamadas al modelo de datos, ya que hay que acceder a tres clases diferentes y relacionarlas.

- No permite guardar correctamente las preguntas de tipo *Translation*, ya que la respuesta no correspondía con el atributo "identificador" (utilizado para comprobar si la respuesta era correcta) de la clase `Item` (en el ejemplo, la pregunta sería "*house*" para que el usuario responda "casa").
- Se considera más sencillo la agrupación de ítems por series que no por conceptos semánticos como representan las listas, y finalmente se observa que para poder calcular el progreso de usuario tan sólo es necesario llevar un control de los ítems de forma individual.

6.7.3. Implementación definitiva del algoritmo

Poco a poco, el modelo de datos ha ido cambiando hasta llegar al modelo que se ha implementado. La principal diferencia, además de algunos nuevos atributos, es que un problema ahora representa una única pregunta, y no un conjunto de preguntas.

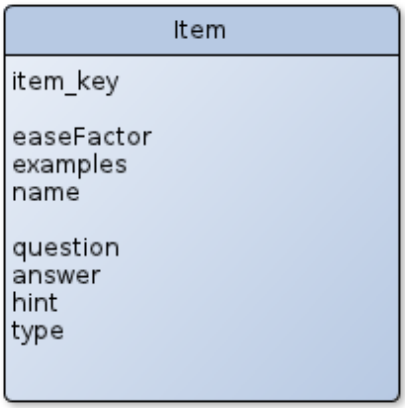


Figura 6.13: Clase `Item`.

Hay dos implicaciones importantes a resaltar con esta nueva aproximación: la primera es que se trata de una representación más flexible porque incluye un atributo para la pregunta y otro para la respuesta, y ambos pueden ser tan largos o cortos como sea necesario; la segunda es que ya no es obligatorio que un a un mismo problema se le relacionen múltiples preguntas (de más de un tipo). Este nuevo enfoque, además, permite acceder a todos los problemas de una misma sesión de estudio de una manera mucho más rápida y sencilla.

Comparando esta aproximación con la inicial se puede observar cómo soluciona los tres principales inconvenientes explicados en el apartado 6.7.2: sólo son necesarias cinco instancias para almacenar cinco problemas, por lo que son necesarias menos consultas al modelo de datos y, además, son todas instancias de la misma clase. El problema del tipo de pregunta *Translation* queda resuelto y ampliamente mejorado por haber añadido un atributo que almacene la pregunta y otro que almacene la respuesta.

El siguiente ejemplo muestra un hipotético ejercicio válido para la palabra inglesa *house* con los mismos cinco problemas del apartado 6.7.2 (los valores de los atributos "identificador" son valores inventados para este ejemplo y sólo se muestran los atributos relevantes para la comparación):

Clase `Item`:

- identificador: Item1
  - `question`: (Ruta al fichero de audio)
  - `answer`: house
  - `type`: Audio
- identificador: Item2
  - `question`: (Ruta al fichero de imagen)
  - `answer`: house
  - `type`: Image
- identificador: Item3
  - `question`: /haʊs/
  - `answer`: house
  - `type`: Phonetics
- identificador: Item4
  - `question`: casa
  - `answer`: house
  - `type`: Translation
- identificador: Item5
  - `question`: house
  - `answer`: casa
  - `type`: Word

6.7.4. Las sesiones de estudio

Una vez que el usuario se ha suscrito a algún curso puede empezar a estudiar su contenido, que se ha estructurado en series, tal y como se ha explicado en el apartado 3.2.1.

Una vez que el usuario ha accedido a su página de inicio en el sistema, puede estudiar uno de los cursos a los que se ha suscrito. A partir de aquí, el sistema le mostrará los problemas a resolver siguiendo este algoritmo:

1. En primer lugar se muestran los problemas realizados anteriormente que estén pendientes de mostrar según la planificación, siempre que pertenezcan al curso actual. Esto implica, por tanto, que han de ser problemas de series ya finalizadas o bien interrumpidas.

2. En segundo lugar se muestra la primera serie no realizada de ejercicios del curso actual; es decir, el sistema determina en qué punto del curso está el usuario.

3. Al finalizar los problemas anteriores:

- Si hay más series en el curso, se le ofrece al usuario estudiar la siguiente serie.
- Si no hay más series en el curso, se le indica al usuario que ha finalizado todas las series del curso actual. En el futuro puede volver a revisar los problemas según la planificación efectuada por el sistema. Esto significa, por tanto, que el estudio queda en manos de la planificación realizada por el sistema.

Cualquier problema que sea respondido de manera equivocada se repetirá durante la misma sesión de estudio hasta que se responda de manera correcta. El usuario puede hacer servir la ayuda para el ejercicio siempre que lo necesite. Ver apartado 6.7.5 para más información sobre la planificación de los ejercicios conforme se van resolviendo.

Durante la sesión de estudio, cada problema presentado muestra algunos datos del progreso en dicho problema, si es que se había estudiado alguna vez con anterioridad. En caso contrario, se indica que es la primera vez que se presenta el problema al usuario y se muestran los ejemplos de uso para un mejor aprendizaje, si los hubiera.

Cuando el usuario introduce una respuesta y le da al botón para pasar al siguiente ejercicio, verá al lado del campo donde ha introducido la respuesta una de las siguientes opciones:

- Si la respuesta es correcta, sobre un fondo verde aparece un ✔.
- Si la respuesta no es correcta, sobre un fondo rojo aparece una ✖.

Por otro lado, y de manera independiente, cada problema tiene asociado un tiempo en segundos que indica el tiempo máximo esperado para resolver el problema. La ventana de estudio muestra una barra de progreso que mide el tiempo y decrece a medida que pasa hasta que ya no queda más tiempo. En ese instante aparece un aviso en la parte superior que indica que, pese a que el tiempo ha pasado, aún puede responder a la pregunta. La única penalización que este hecho implica es que la `quality_of_the_response` del usuario respecto al problema será inferior según el tiempo extra empleado (más información sobre esta variable en el apartado 6.7.5.1).

6.7.5. Algoritmo de repetición espaciada en el tiempo

El algoritmo implementado en este proyecto y que utiliza la técnica de la repetición espaciada en el tiempo de ejercicios se detalla a continuación:

- El atributo `ease_factor` se inicializa a 2.5
- La función para calcular el número de días hasta la siguiente repetición del ítem es la misma que utiliza el `SM2`.
- Igual que en *Anki*, si la respuesta es errónea se repite indefinidamente en esa sesión de estudio.
- En caso de que el problema se esté estudiando por primera vez, se planifica su repetición al día siguiente pero sin actualizar el valor del atributo `ease_factor`. De la misma forma, existe una cola de aprendizaje y otra de retención en la que se almacenan los problemas según la fase de estudio en la que se encuentren para el usuario.

- Si una tarjeta se ha resuelto "fácilmente", el `ease_factor` puede aumentar más allá de 2.5.

- La valoración que se hace de la respuesta del usuario tiene en cuenta el tiempo de resolución. Dependiendo del valor obtenido se modifica convenientemente el atributo `quality_of_the_response`.

- A partir del cálculo del valor de `quality_of_the_response` se puede planificar el problema adecuadamente y sin la intervención del usuario.

La planificación de un problema determinado se realiza en el momento en que el usuario desea avanzar al siguiente problema (en caso que hubiera). Justo después de ese momento, el sistema comprueba si la respuesta dada es correcta o no y comienza el proceso que determinará cuándo el usuario debe volver a estudiar el problema. De ello se encarga la función `update()` de la clase `ProblemSolved` que, a su vez, llama a la función privada `schedule()` para este menester.

La próxima vez que el usuario vuelva a utilizar el servicio e indique que quiere continuar estudiando un curso, el sistema determinará si tenía problemas pendientes de estudio dependiendo de la fecha en la que estuvieran planificados respecto la fecha actual. En caso de que el usuario tuviera problemas "atrasados", éstos serán mostrados al inicio de la sesión de estudio, tal y como se indicó en el apartado 6.7.4.

6.7.5.1. Atributos esenciales para la planificación de ejercicios

Existen ciertos parámetros muy importantes a la hora de realizar la planificación de un ejercicio resuelto. Estos parámetros son representados por los siguientes atributos de la clase `problemSolved`: `ease_factor`, `quality_of_the_response` y `queue`.

Cada uno de estos atributos juega un papel muy importante a la hora de determinar la siguiente repetición del problema al usuario. A continuación se da una explicación detallada de la utilidad de cada uno de ellos.

`ease_factor` Este atributo representa el nivel de dificultad de un problema. Inicialmente, su valor por omisión se establece a **2.5** en el cálculo la primera vez que el usuario resuelve un problema. Dicho valor se va actualizando de forma automática según cada usuario y las respuestas que éste provee en sus sesiones de estudio. Este valor inicial de 2.5 indica la cualidad de "fácil" para todos los problemas. A medida que el usuario va resolviendo problemas, se actualiza a partir de otro atributo esencial llamado `quality_of_the_response` de la siguiente manera:

$EF' := EF + (0{,}1 - (5 - q) * (0{,}08 + (5 - q) * 0{,}02))$

"EF" es `ease_factor`

"q" es `quality_of_the_response`

En el momento en que la calidad de la respuesta no es adecuada (tal y como se explica en el apartado 6.7.5.1), el `ease_factor` puede llegar a descender hasta un valor mínimo de **1.3**, denotando así que ese problema resulta muy difícil de resolver para el usuario. De esa forma, cuanto más difícil es recordar un concepto, mayor es el decremento en el valor del `ease_factor`.

Sin embargo, es necesario mencionar que dicho valor nunca será menor de 1.3 ya que entonces el problema se considera de una dificultad muy elevada para el usuario y conlleva unas repeticiones excesivamente cercanas en el tiempo[15].

Finalmente, destacar que este atributo también se actualiza en función de la fase de estudio en la que se encuentre el usuario. Esto es debido a que no se puede penalizar de la misma forma a un usuario si es la primera vez que está estudiando el problema, o si, por el contrario, ya lo ha estudiado con anterioridad. Esto se explica con más detalle en el apartado del atributo `queue`.

`quality_of_the_response` Este atributo se refiere a cómo de bien ha respondido el usuario una pregunta planteada en un problema. Los aspectos que se tienen en cuenta a la hora de validar la calidad de una respuesta son los siguientes:

- Tiempo máximo para resolver un problema.
- Tiempo en el que se considera que el ítem se domina calculado por la función `get_time_to_master()` de la clase `Item`.

Los valores que puede tener este atributo se sitúan en un rango de [1, 4]. Tras la realización de diversos tests descritos en las fases de prueba en el apartado 7.1 (en total, cuatro aproximaciones o iteraciones de resolución), se ha determinado que con esos cuatro valores se puede realizar una asignación de valor a dicho atributo de una forma coherente con la realidad; es decir, que afecta de forma lógica al cálculo del valor del `ease_factor`. Por ejemplo: a mayor tiempo de respuesta, la calidad de la respuesta será peor (con un valor bajo, de 1 ó 2) y por lo tanto, el valor del `ease_factor` decrecerá proporcionalmente.

Estos valores se han tomado a partir de las dos aproximaciones que realizan los algoritmos implementados en *Supermemo* y en *Anki*.

En contraposición a lo que se ha comentado de las dos aplicaciones anteriores, este atributo se calcula de manera automática, sin requerir en ningún caso la valoración de la pregunta por parte del usuario haciendo, así, del estudio una tarea más centrada en analizar el problema, resolverlo y pasar al siguiente. Una tarea, por tanto, **mucho más dinámica en este sentido**.

`queue` Este atributo representa la fase de aprendizaje en la cual se encuentra un problema determinado para un usuario. Aunque *Supermemo* no implementa esta variante en su versión `SM2` del algoritmo, *Anki* sí lo hace y sin embargo no es un concepto novedoso (tal y como se explica en el apartado 2.1).

Se ha decidido relacionar el concepto de las cajas Leitner (método alternativo) explicado en el apartado 2.1 a listas de elementos.

Como se puede ver comparando las figuras 2.1 y 2.2, en el sistema diseñado en este proyecto se determina que un problema respondido erróneamente pasa a una lista anterior y no a la primera lista. Estas listas se agrupan en dos colas que representan el estado en el que se encuentra el problema que ha resuelto el usuario. De esta manera se tratan de forma diferente aquellos problemas que se están resolviendo por primera vez (fase de aprendizaje) respecto de aquellos problemas estudiados con anterioridad (fase de retención)

Así, los problemas en fase de aprendizaje se planifican de manera diferente a los problemas en fase de retención en caso de error: los primeros serán repetidos de forma más

[15]El autor explica cómo sus observaciones indicaban que no era aconsejable bajar de este valor: `http://www.supermemo.com/english/ol/sm2.htm` .

cercana en el tiempo y los segundos al contrario. Además, serán movidos a la caja inmediatamente anterior y, por lo tanto, tendrán una planificación adecuada al estado en que se encuentren teniendo en cuenta que se han fallado recientemente, pero también que son problemas que ya se han estudiado en anteriores ocasiones.

6.7.5.2. Guardar los datos de la respuesta

En la figura 6.14 se muestra un esquema del algoritmo utilizado para guardar los datos de la respuesta que proporciona el usuario en el proceso de estudio.

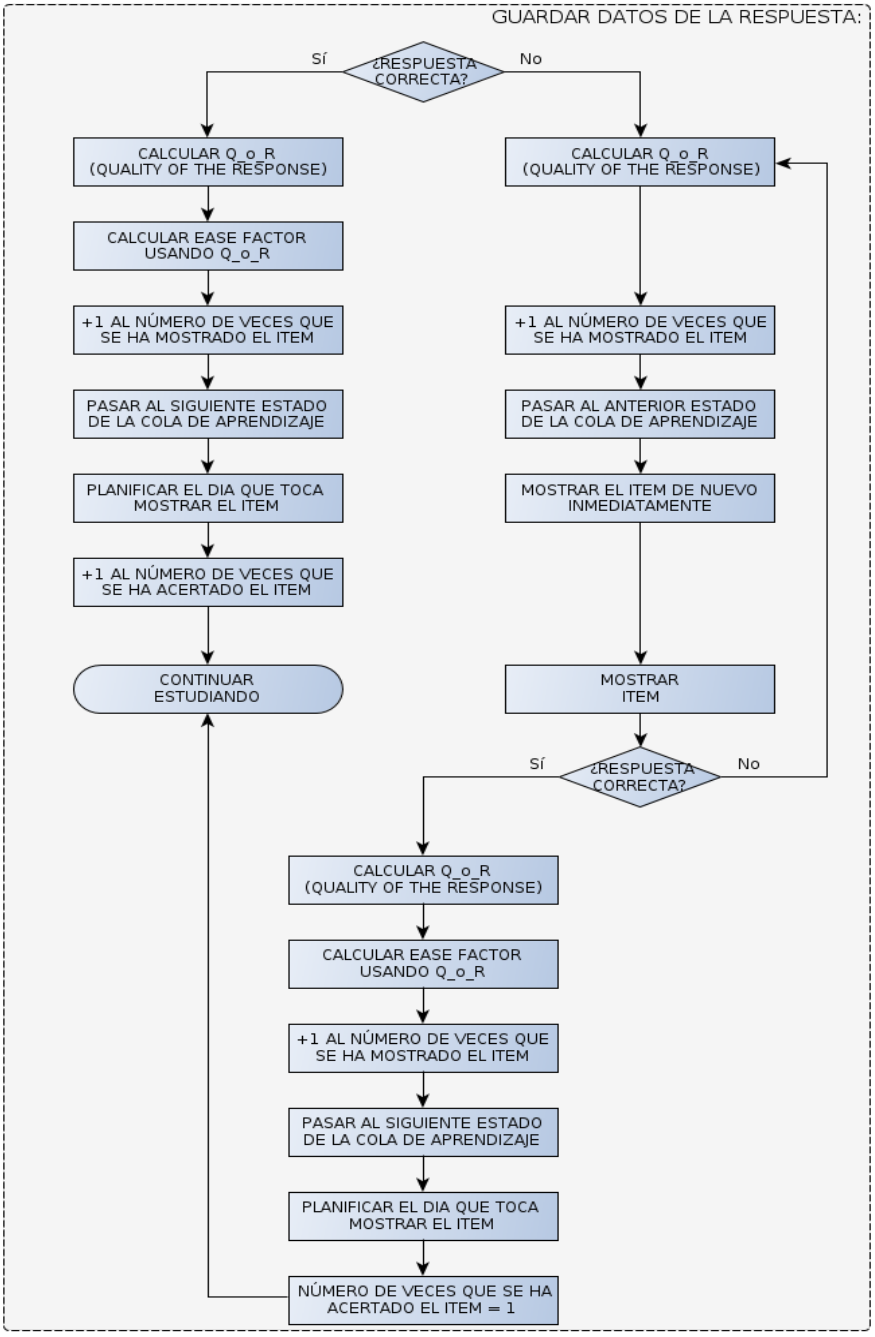


Figura 6.14: Algoritmo utilizado para guardar los datos de la respuesta del usuario.

Capítulo 7

Transición del sistema

El objetivo principal de la fase de transición es asegurar que el producto llega a los usuarios finales, y que éstos lo pueden utilizar, es decir, el paso hacia la puesta en marcha. Pese a que se trata de una fase que generalmente se divide en algunas iteraciones, para un proyecto de esta envergadura no se ha creído conveniente.

Esta fase abarca los siguientes puntos:

- Última etapa de pruebas: pruebas realizadas por terceras personas ajenas al desarrollo de la aplicación.
- Detección y aplicación de cambios: a raíz de las pruebas, se detecta y corrige qué es necesario cambiar en el proyecto.
- Definición de la versión final del producto.
- Documentación del proyecto: unificar los documentos creados gracias a `RUP` en una memoria y realización de la documentación de esta última fase, que incluye las últimas pruebas así como el manual del desarrollador y del usuario.
- Revisar de forma global la coherencia entre el proyecto desarrollado y la documentación generada antes de la entrega.

7.1. Etapa de pruebas final

La etapa de pruebas de esta fase se subdivide en dos: las pruebas realizadas por terceras personas y las pruebas que confirman que los cambios se han aplicado correctamente. Esta etapa de pruebas es esencial, pues es necesaria para poder definir una versión definitiva del proyecto.

Se pidió a varias personas ajenas al proyecto que probaran exhaustivamente la aplicación, reportaran los errores encontrados e hicieran llegar cualquier sugerencia que pudiera repercutir en una mejor aplicación final. Estas pruebas han sido muy importantes pues han permitido detectar errores que por el hecho de estar desarrollando y probando frecuentemente la aplicación ya habían pasado a formar parte de "lo habitual" y no se había definido como algo erróneo en un primer momento. Además, ciertas sugerencias como por ejemplo el poder disponer de más tiempo para resolver ciertas preguntas más difíciles que otras o el dar información del proceso de estudio por primera vez (a pesar de pensar que es suficientemente autoexplicativo) ha permitido mejorar la aplicación resultante de forma sustancial.

El ciclo de las etapas de pruebas realizadas en todo el proyecto siempre han seguido estos pasos en mayor o menor medida:

1. Definir un punto de inflexión en la programación y decidir, de acuerdo a la planificación, el momento adecuado para realizar pruebas.

2. Comenzar una batería de pruebas de aquello que se ha programado apuntando en una lista todo lo que es necesario corregir.

3. Realizar cada cambio de forma atómica, es decir: escoger un cambio a realizar, programarlo y volverlo a probar.

4. Una vez que todos los cambios han sido programados y probados, es necesario revisar la *web* en su conjunto y comprobar que todo funciona correctamente. En caso contrario, en necesario realizar otra iteración de pruebas volviendo al punto número 2.

7.2. Detección y aplicación de cambios

A raíz de todas las etapas de pruebas realizadas a lo largo del desarrollo se han ido generando cambios y mejoras y sobre todo corrigiendo errores importantes para el buen funcionamiento de la aplicación. Sin embargo, es en esta última fase donde los cambios han de estar mejor acotados, puesto que el desarrollo ha concluido en la fase anterior y en esta última fase de transición estos cambios han de significar tan sólo la corrección de errores que han pasado desapercibidos previamente o alguna mejora que no suponga un cambio mayor en el código de la aplicación.

Cualquier otro cambio que no cumpla con las características mencionadas, deberá pertenecer a una lista de cambios o mejoras para una segunda versión de la aplicación presentada en este proyecto. La lista de cambios que formarían parte de una segunda versión del proyecto se puede consultar en el anexo A.5.

7.3. Documentación desarrollada

Se entiende por "documentación" cualquier material utilizado para explicar detalles y atributos de un sistema o de sus partes. En términos de ingeniería, la documentación de una aplicación suele ser material impreso o en formato electrónico que describe la estructura, los componentes y las operaciones de un sistema o producto. Pese a ser menospreciada muchas veces por ser la parte "aburrida" de la ingeniería, o incluso considerada en ocasiones superflua o innecesaria, lo cierto es que es una parte muy importante de todo desarrollo.

Los tipos de presentación más frecuentes son la ayuda en línea, los manuales y las guías de usuarios.

El propósito del manual para los usuarios finales de la aplicación, es explicar cómo funciona el sistema y cómo utilizarlo. La documentación realizada para ellos está definida en el apartado 2.8.2.

Por otra parte, el propósito del manual del desarrollador es indicar cómo instalar el sistema, documentar su estructura interna, crear nuevo contenido para el estudio así como dar a conocer los problemas existentes en la versión actual. Se puede encontrar en el anexo A.

Capítulo 8

Conclusiones y valoración personal

Una vez finalizado el proyecto es momento de realizar un repaso global de todo el proceso, desde la idea inicial hasta el producto que se ha desarrollado, para poder sacar conclusiones y realizar una valoración de aquello que se ha conseguido y aquello que no.

8.1. Conclusiones

Objetivamente hablando, **se han conseguido los objetivos principales** definidos en el apartado 2.3. En resumen, se ha construido un sistema que cumple con lo especificado en el alcance del proyecto: una aplicación que permite a cualquier usuario realizar un estudio espaciado en el tiempo de una determinada materia, que planifica correctamente los ejercicios para cada usuario en función del progreso en cada uno de los problemas resueltos, que tiene unas mínimas cualidades de usabilidad y diseño y que es ampliable de forma fácil, tanto en contenido como en funcionalidades siguiendo el manual del desarrollador (anexo A).

Salvo determinadas funcionalidades explicadas en el apartado 6.2.1, se ha implementado el sistema prácticamente en su totalidad según la planificación establecida inicialmente, dando lugar a un sistema plenamente funcional.

A pesar de esto, el sistema se puede mejorar (como se explica en el apartado A.5) y ampliar en cuanto a funcionalidad, dando opción a realizar una segunda versión que incluya los casos de uso de prioridad "media-baja" o "baja" que no han tenido cabida finalmente en este proyecto.

8.2. Valoración personal

La valoración personal después de la realización del proyecto es positiva. El hecho de tener que iniciar un proyecto desde cero, sin más referencias que la existencia de otras aplicaciones similares en el mercado, pero con la motivación añadida de intentar mejorar esa oferta existente, es sin duda un hecho del que me siento orgullosa. El camino no ha sido fácil pues ha sido necesario aprender desde el principio el lenguaje de programación (en este caso *Python*, haciendo uso de bibliografía muy útil [1]), aprender el funcionamiento de *Google App Engine* [11], profundizar en el diseño de la arquitectura que propone `REST` [16] o aprender otros lenguajes como `HTML`, `CSS` o *JavaScript*. Estos requisitos, y muchos otros, han hecho que la carga de trabajo fuera considerable (algo más de un año de trabajo), pero que, sin duda, haya merecido la pena al final.

Utilizar la aplicación final obtenida y observar que, haciendo los estudios de mercado pertinentes, podría llegar a tener una salida comercial o sencillamente pedagógica, es signo de que se ha creado un producto con una calidad aceptable y que puede llegar a ser **utilizado**.

Para concluir, me gustaría añadir que he aprendido que en la realización de un proyecto final de carrera es donde se pone de manifiesto que los conceptos aprendidos en la carrera son una buena base para empezar a trabajar, pero que **siempre** será necesario continuar aprendiendo si se quiere cumplir con las metas que uno se marca al inicio.

Apéndice A

Manual del desarrollador

Con este manual se pretende realizar una introducción guiada a la mejora del sistema y a la creación de nuevos cursos para el estudio. El sistema está programado íntegramente en *Python*, haciendo uso de la orientación a objetos donde las clases tienen una estructura bien definida y clara.

A.1. El entorno de trabajo

El entorno de trabajo habitual para el desarrollo de este proyecto ha sido `GNU/Linux` con un entorno de escritorio `KDE 4` y la distribución *Ubuntu 11.10* (*Oneiric Ocelot*).

A.2. La primera vez que se crea una cuenta en GAE

Tras entrar en *Google App Engine* y registrar la cuenta, se recibe un `sms` con el código de activación de la cuenta para empezar a desarrollar las aplicaciones que se deseen. Se dispone de hasta 10 aplicaciones por cuenta. Una vez hecho esto se pide crear la primera aplicación. Cuando se ha creado la aplicación, se puede acceder a ella mediante la `URL` que aparece en los menús de configuración de *Google App Engine* (`https://appengine.google.com/`). En ese momento, es recomendable programar algo sencillo para realizar así alguna prueba y comprobar el funcionamiento.

A.3. Cómo continuar con el desarrollo del proyecto

Para poder realizar una continuación del proyecto es necesario seguir los pasos detallados a continuación. Se asume una instalación de un sistema operativo `GNU/Linux` como *Ubuntu*:

1. Instalar las siguientes aplicaciones:
   a) *Python* versión 2.7.
   b) Un entorno de desarrollo integrado como *Eclipse*.
   c) El sistema de control de versiones *Mercurial*.
   d) El entorno de desarrollo de *Google App Engine*.
2. Instalar *PyDev* desde *Eclipse*:

   a) Ir al menú "*Help → Install New Software...*" y añadir el repositorio siguiente: `http://www.PyDev.org/updates` .

   b) Instalar *PyDev* y continuar con los pasos del asistente de instalación.

3. Reiniciar *Eclipse*. Configurar *Google App Engine* en *Eclipse*:

   a) Ir al menú "*Window → Preferences → PyDev → Interpreter - Phyton*" y apretar el botón "*Auto Config*".

   b) Una vez allí, es necesario asegurarse de que el intérprete de *Python 2.7*, como mínimo, está seleccionado (generalmente `/usr/lib/python2.7`) (más información en el paso 16).

   c) Reiniciar *Eclipse*.

4. Crear bajo el directorio personal una carpeta de nombre *workspace*.

5. Descargar el código del proyecto desde su repositorio:

   a) `cd ~/workspace`

   b) `hg clone http://bitbucket.org/ana_hernandez/ulearn`

6. Crear un nuevo proyecto:

   a) Ir al menú "*File → New → Project...*" y seleccionar "*PyDev → PyDev Google App Engine Project*".

   b) Una vez allí, es necesario rellenar un formulario con estos datos:

      i *Project name: ulearn*

      ii *Application id registered: ulearnpfc*

      iii *Template: Empty project*

7. Descargar *Bootstrap*[1] y copiar las carpetas *css*, *js* e *img* a la carpeta *workspace* de manera que existan:

   a) *~/workspace/ulearn/css/*.css*

   b) *~/workspace/ulearn/img/*.png*

   c) *~/workspace/ulearn/js/*.js*

   d) *~/workspace/ulearn/js/bootstrap-tooltip.js*

8. Descargar otro fichero `CSS` de *Bootstrap*[2] y guardarlo en la carpeta *css* tal que:

   a) *~/workspace/ulearn/css/docs.css*

9. Descargar la clase *gen_validatorv4.js*[3] y copiarla en *~/workspace/ulearn/js*.

10. Realizar las siguientes instrucciones:

    a) `cd ~/workspace/ulearn/js`

[1] *Bootstrap* principal: `http://twitter.github.com/bootstrap/` .

[2] *Bootstrap docs:* `http://twitter.github.com/bootstrap/assets/css/docs.css` .

[3] *JavaScript Form Validator:* `http://www.javascript-coder.com/html-form/javascript-form-validation.phtml` .

b) `ln -s gen_validatorv4.js form_validator.js`

11. Descargar la última versión de *jQuery-UI*[4].

12. Realizar las siguientes instrucciones escogiendo el tema "*smoothness*":

    a) `unzip jquery-ui-1.9.0.custom.zip`

    b) `cp jquery-ui-1.9.0.custom/js/jquery-ui-1.9.0.custom.min.js ~/workspace/ulearn/js/jquery-ui-1.9.0.custom.min.js`

    c) `cp jquery-ui-1.9.0.custom/js/jquery-1.8.2.js ~/workspace/ulearn/js/jquery-1.8.2.js`

    d) `cp -r jquery-ui-1.9.0.custom/css/<TEMA_ESCOGIDO> ~/workspace/ulearn/css/`

    e) `cd ~/workspace/ulearn/js`

    f) `ln -s jquery-1.8.2.js jquery.js`

    g) `ln -s jquery-ui-1.9.0.custom.min.js jquery-ui.js`

    h) `cd ~/workspace/ulearn/css/<TEMA_ESCOGIDO>`

    i) `ln -s jquery-ui-1.9.0.custom.min.css jquery-ui.css`

13. Descargar la última versión de *Less*[5].

14. Realizar las siguientes instrucciones:

    a) `ln -s less-1.3.0.min.js less.js`

15. Forzar a que el módulo *memcache* sea un "*forced builtin*" de *Python* dentro del proyecto:

    a) Ir al menú "*Window* → *Preferences* → *PyDev* → *Interpreter Python*".

    b) Acceder a la pestaña "*Forced Builtins*" y luego a "*New*".

    c) Una vez allí, es necesario añadir "*google.appengine.api.memcache*"

    d) Salir aplicando los cambios.

    e) Reiniciar *Eclipse*.

16. Instalar algunos módulos externos dentro de la carpeta *workspace*:

    a) Descargar *python-pagination*[6] y extraer su contenido. Copiar en *~/workspace/ulearn* cambiando el nombre de la carpeta que contiene el módulo por *python_pagination*.

    b) Descargar *appengine-rest-server*[7] y extraer su contenido. Copiar en *~/workspace/ulearn* tan sólo la carpeta *rest* con el archivo `__init__.py`.

    c) Descargar *gae-sessions*[8] y extraer su contenido. Copiar en *~/workspace/ulearn* cambiando el nombre de la carpeta por *gaesessions* con el archivo `__init__.py`.

[4] *jQuery:* `http://jqueryui.com/download/` .
[5] *Less:* `http://lesscss.org/` .
[6] *python-pagination*: `http://code.google.com/p/python-pagination/` .
[7] *appengine-rest-server*: `http://code.google.com/p/appengine-rest-server/` .
[8] *gae-sessions*: `https://github.com/dound/gae-sessions` .

d) Descargar la versión más reciente de la rama 1.x del módulo *python-dateutil*[9]. Copiar la carpeta *dateutil* entera en *∼/workspace/ulearn*.

e) Reiniciar *Eclipse* y comprobar que las importaciones de los módulos son correctas.

17. Para comenzar a programar y poder observar los cambios que se van realizando, es necesario ejecutar el servidor de desarrollo:

    a) `∼/google_appengine/dev_appserver.py ∼/workspace/ulearn --debug --default_partition=“”`

18. Para subir la aplicación a producción es necesario ejecutar la siguiente instrucción. Es posible cambiar tanto la versión de la aplicación[10] modificando convenientemente el fichero `app.yaml` teniendo en cuenta que esto incurrirá en una nueva versión disponible en la consola de administración de *Google App Engine*.

    a) `∼/google_appengine/appcfg.py update ∼/workspace/ulearn`

A.4. Ampliando el sistema

Es muy fácil añadir nuevo material al sistema: es necesario iniciar sesión como un usuario con privilegios para acceder al menú de administración de la *web*, que permite, entre otras opciones, crear los problemas, luego las series y, por último, asociar las series a un curso (sea de nueva creación o no).

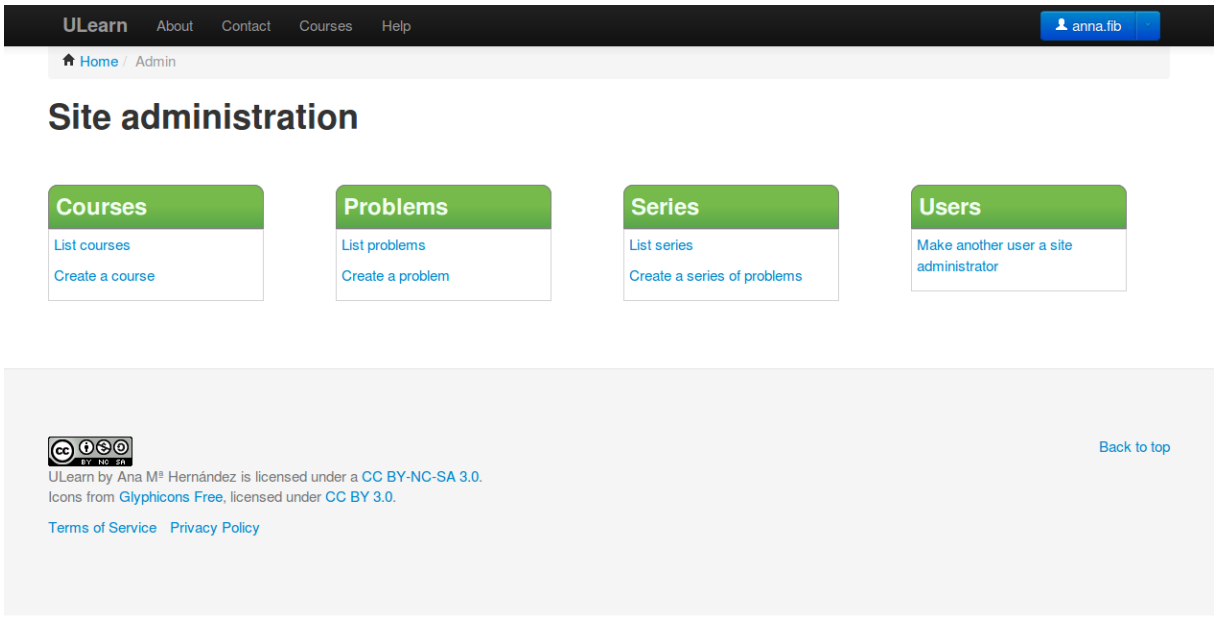


Figura A.1: Página inicial de la administración del sistema.

[9] *python-dateutil*: `http://labix.org/python-dateutil/` .

[10] Más información en: `https://developers.google.com/appengine/docs/python/config/appconfig` .

No obstante, también es posible crear nuevos tipos de preguntas modificando mínimamente el código de la aplicación, tal y como se verá en el apartado siguiente. De manera similar se listan posibles mejoras que realizar de cara a la próxima versión del sistema.

A.4.1. Extendiendo los tipos de pregunta posibles

El sistema se ha diseñado de manera que es muy fácil añadir nuevos tipos de pregunta como, por ejemplo, definiciones a las que hay que responder con la palabra definida (este caso es parecido al tipo "*Word*"). El proceso que hay que seguir para que el sistema acepte estos nuevos tipos son los siguientes:

1. Añadir el nuevo tipo al resultado retornado por la función `get_item_types()` de la clase `Item`.
2. Añadir el nuevo tipo a la función `to_template()` de la clase `Item`.
3. Añadir el nuevo tipo al atributo `type` de la clase `Item`.
4. Crear la plantilla para mostrar al usuario la pregunta similarmente a las que ya hay en la carpeta *templates/item/*.
5. Indicar a la plantilla que muestra cualquier tipo de problemas (fichero "*templates/solving_problems.html*") qué plantilla usar si el tipo de ejercicio es el que se ha añadido.

Después de estos pasos, sólo es necesario crear las preguntas pertinentes, crear un nuevo curso si es necesario y crear las nuevas series con los ejercicios.

A.5. Mejoras para la próxima versión

Se han detectado algunas mejoras que podrían implementarse en la siguiente versión del sistema; a continuación se detallan en orden de importancia.

- Penalizar el uso de la ayuda en la resolución de problemas, así como si una serie no es acabada en la misma sesión de estudio en que es empezada.
- Mostrar el progreso del estudio del curso de un usuario al finalizar una serie. Ahora tan sólo se muestra el progreso obtenido de una serie justo al finalizarla, no del curso al completo.
- Implementar los casos de uso de la gestión de material aportado por los usuarios, cuya baja prioridad los ha excluido de la versión actual del sistema.
- Durante la resolución de un ejercicio, se podría detener el tiempo en el intervalo de escritura de la respuesta y volver a ponerlo en marcha cuando se detecte que no se están pulsando más teclas.
- De nuevo, durante la resolución de ejercicios se podría añadir un botón de pausa para que el sistema no muestre el siguiente problema hasta que lo indique el usuario. De esta manera se evitan las posibles trampas si el usuario pausara un problema todavía no resuelto.

- Permitir hacer *login* con las otras dos vías de autenticación soportadas por el servicio de *Google*[11].
- Tratamiento específico de palabras homófonas permitiendo visualizar al usuario un ejemplo de uso de la palabra sin que lo haya solicitado y sin que afecte al cálculo de su progreso.

[11] Más información en: `https://developers.google.com/appengine/docs/python/users/` .

Apéndice B

Planificación y diagramas de *Gantt*

En este anexo se muestra la planificación de cada fase del proyecto que permite asignar los recursos necesarios para un correcto desarrollo de las tareas que se realizan en cada fase e iteración.

Es útil planificar una fase con un diagrama de *Gantt*, ya que ofrece una visión rápida y global de las tareas a realizar, sus requisitos previos y la estimación de su duración, que se mide en días y se refleja en una fecha de inicio y de final sobre un calendario. Es necesario considerar que en el diagrama no aparecen las personas responsables de cada tarea o acción, ya que el proyecto es desarrollado por un único miembro, tal y como se indica en el apartado 2.5.3.

B.1. Fase inicial

El objetivo principal de la fase inicial es describir el contexto del sistema, el modelo de negocio y trazar las bases del modelo de casos de uso del sistema.

B.1.1. Planificación temporal mediante un diagrama de *Gantt*

B.1.1.1. Planificación previa

Se ha determinado que sólo es necesaria una iteración en esta fase, que comenzará el lunes 28 de noviembre de 2011 y durará hasta el viernes 10 de febrero de 2012.

B.1.1.2. La realidad de la fase inicial

Pese a la planificación del apartado anterior, la realidad es que la planificación de esta fase se vio ligeramente retrasada por la identificación y documentación de los riesgos del proyecto y por la definición de los casos de uso. El retraso total con el que se ha finalizado esta fase es de siete días (once días naturales).

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Inicial**	**nov 28**	**feb 10**	**51d**
1.1	**Iteración I1**	**nov 28**	**feb 10**	**51d**
1.1.1	Iniciar Proyecto	nov 28	nov 28	N/D
1.1.2	**Documentación y preparación del entorno de trabajo**	**nov 28**	**ene 27**	**41d**
1.1.2.1	Estudios previos entorno de trabajo	nov 28	dic 2	5d
1.1.2.2	Prueba entorno de trabajo	dic 5	dic 9	3d
1.1.2.3	Análisis funcional de la aplicación	dic 12	dic 12	1d
1.1.2.4	Documentación de la metodología RUP	dic 13	dic 14	2d
1.1.2.5	Documentación del objetivo	dic 15	dic 20	4d
1.1.2.6	Documentación del alcance	dic 21	dic 28	5d
1.1.2.7	Documentación de la estrategia	dic 29	ene 4	5d
1.1.2.8	Documentación de los requisitos	ene 5	ene 17	8d
1.1.2.9	Documentación de los riesgos	ene 18	ene 27	8d
1.1.3	**Especificación inicial**	**ene 30**	**feb 10**	**10d**
1.1.3.1	Primer borrador de los casos de uso	ene 30	feb 10	10d
1.1.4	Entrega parcial de la documentación	feb 10	feb 10	N/D

Figura B.1: Fechas de la fase inicial.

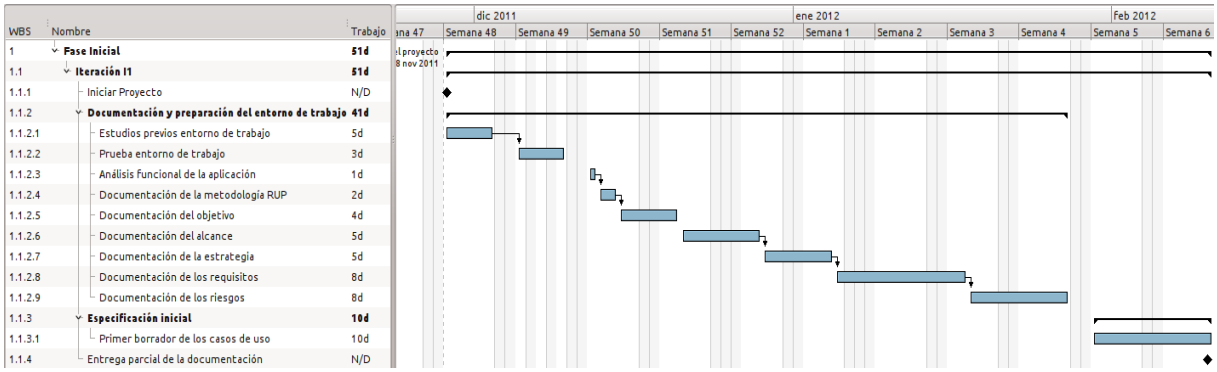


Figura B.2: Distribución temporal de la fase inicial.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Inicial**	**nov 28**	**feb 21**	**58d**
1.1	**Iteración I1**	**nov 28**	**feb 21**	**58d**
1.1.1	Iniciar Proyecto	nov 28	nov 28	N/D
1.1.2	**Documentación y preparación del entorno de trabajo**	**nov 28**	**ene 31**	**43d**
1.1.2.1	Estudios previos entorno de trabajo	nov 28	dic 2	5d
1.1.2.2	Prueba entorno de trabajo	dic 5	dic 9	3d
1.1.2.3	Análisis funcional de la aplicación	dic 12	dic 12	1d
1.1.2.4	Documentación de la metodología RUP	dic 13	dic 14	2d
1.1.2.5	Documentación del objetivo	dic 15	dic 20	4d
1.1.2.6	Documentación del alcance	dic 21	dic 28	5d
1.1.2.7	Documentación de la estrategia	dic 29	ene 4	5d
1.1.2.8	Documentación de los requisitos	ene 5	ene 17	8d
1.1.2.9	Documentación de los riesgos	ene 18	ene 31	10d
1.1.3	**Especificación inicial**	**feb 1**	**feb 21**	**15d**
1.1.3.1	Primer borrador de los casos de uso	feb 1	feb 21	15d
1.1.4	Entrega parcial de la documentación	feb 21	feb 21	N/D

Figura B.3: Fechas reales de la fase inicial.

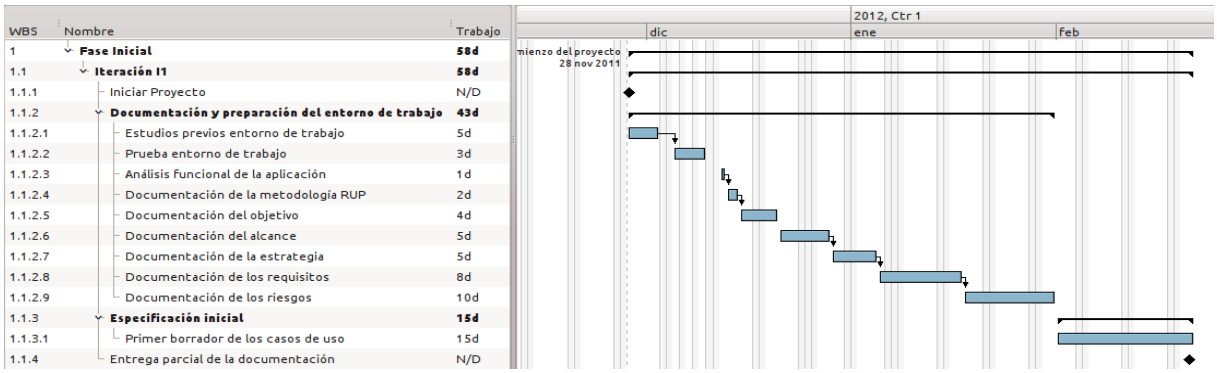


Figura B.4: Distribución temporal real de la fase inicial.

B.2. Fase de elaboración

El objetivo principal de la fase de elaboración es representar de manera abstracta el sistema que se implementará posteriormente, definiéndolo con un nivel de detalle tal que permita su implementación.

B.2.1. Planificación temporal mediante un diagrama de *Gantt*

B.2.1.1. Planificación previa

Se establecen dos iteraciones para esta fase. La primera iteración se realizará entre el lunes 13 de febrero de 2012 y el martes 13 de marzo de 2012; la segunda iteración se hará entre el miércoles 14 de marzo y el miércoles 16 de mayo de 2012, tal y como muestran las imágenes B.5 y B.6.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Elaboración**	**feb 13**	**may 16**	**62d**
1.1	**Iteración E1**	**feb 13**	**mar 13**	**22d**
1.1.1	Planificación de la fase de elaboración - E1	feb 13	feb 14	2d
1.1.2	Especificación detallada de los casos de uso	feb 15	mar 13	20d
1.2	**Iteración E2**	**mar 14**	**may 16**	**40d**
1.2.1	Planificación de la fase de elaboración - E2	mar 14	mar 15	2d
1.2.2	Diseño de la arquitectura del sistema	mar 16	mar 22	5d
1.2.3	Diseño de la arquitectura de la información	mar 23	mar 29	5d
1.2.4	Revisión de los casos de uso de la iteración E1	mar 30	abr 10	3d
1.2.5	Especificación definitiva de los casos de uso	abr 11	may 9	20d
1.2.6	Primera aproximación del algoritmo de repetición	may 10	may 16	5d
1.2.7	Preparación de la fase de construcción	may 16	may 16	N/D

Figura B.5: Fechas de la fase de elaboración.

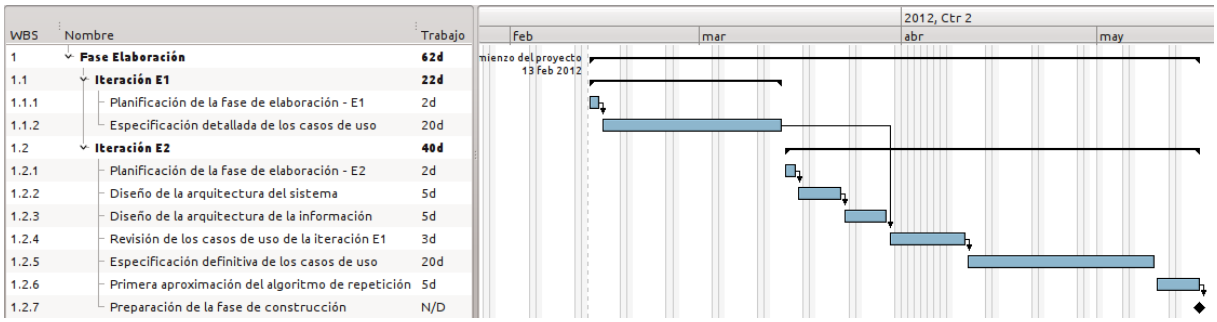


Figura B.6: Distribución temporal de la fase de elaboración.

B.2.1.2. La realidad de la fase de elaboración

La primera iteración finalizó con dos días de retraso respecto a la planificación original, y la diferencia con la entrega de la segunda iteración también fue de dos días. El retraso

total de cuatro días se explica por los casos de uso: se han necesitado más días de los previstos inicialmente para su total redacción y concreción.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Elaboración**	**feb 22**	**jun 1**	**66d**
1.1	**Iteración E1**	**feb 22**	**mar 26**	**24d**
1.1.1	Planificación de la fase de elaboración - E1	feb 22	feb 23	2d
1.1.2	Especificación detallada de los casos de uso	feb 24	mar 26	22d
1.2	**Iteración E2**	**mar 27**	**jun 1**	**42d**
1.2.1	Planificación de la fase de elaboración - E2	mar 27	mar 28	2d
1.2.2	Diseño de la arquitectura del sistema	mar 29	abr 11	5d
1.2.3	Diseño de la arquitectura de la información	abr 12	abr 18	5d
1.2.4	Revisión de los casos de uso de la iteración E1	abr 19	abr 24	4d
1.2.5	Especificación definitiva de los casos de uso	abr 25	may 24	21d
1.2.6	Primera aproximación del algoritmo de repetición	may 25	jun 1	5d
1.2.7	Preparación de la fase de construcción	jun 1	jun 1	N/D

Figura B.7: Fechas reales de la fase de elaboración.

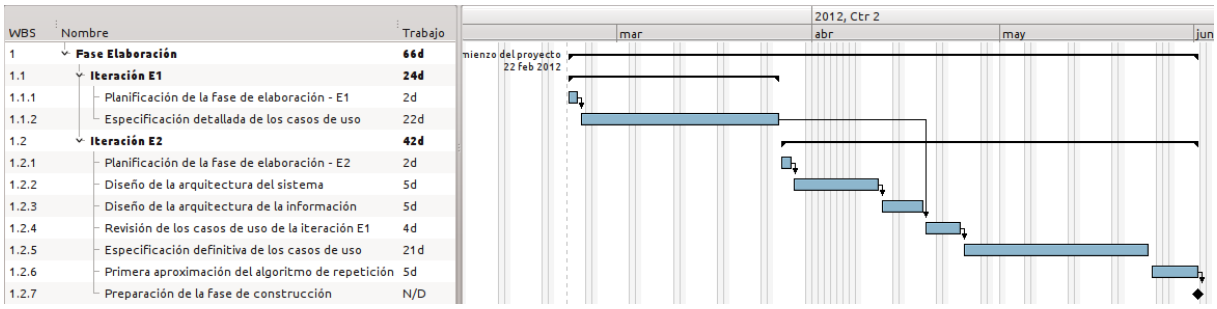


Figura B.8: Distribución temporal real de la fase de elaboración.

B.3. Fase de construcción

El objetivo principal de la fase de construcción es la implementación del sistema diseñado. En consecuencia, al final de las cuatro iteraciones planificadas el sistema debe estar listo para su uso, con más o menos características implementadas según se haya desarrollado esta fase.

B.3.1. Planificación temporal mediante un diagrama de *Gantt*

B.3.1.1. Planificación previa

Se establecen cuatro iteraciones para esta fase, tal y como se ha indicado en el apartado 2.7. La primera iteración se realizará entre el jueves 17 de mayo y el martes 10 de julio; la segunda entre el miércoles 11 de julio y el martes 21 de agosto (con las vacaciones por en medio); la tercera se hará entre el miércoles 22 de agosto y el lunes 27 de agosto; la cuarta y última tendrá que realizarse entre el martes 28 de agosto y el lunes 17 de septiembre de 2012. Estas fechas se muestran visualmente en las imágenes B.9 y B.10.

B.3.1.2. La realidad de la fase de construcción

El retraso total de esta fase es de 21 días de trabajo debido a diversas causas: en la primera iteración el retraso debe atribuirse a una mayor duración de las etapas relativas al progreso del estudio y a la interfaz *web*; en la segunda iteración se debe a la implementación y creación de problemas y a la navegación por el sistema; en la tercera iteración no ha habido ninguna diferencia respecto a lo planificado; finalmente, en la cuarta y última iteración debe atribuirse a la gestión del material de estudio.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Construcción**	**may 17**	**sep 17**	**75d**
1.1	**Iteración C1**	**may 17**	**jul 10**	**38d**
1.1.1	Planificación de la fase de construcción - C1	may 17	may 17	1d
1.1.2	Implementación de la arquitectura REST	may 18	jun 8	15d
1.1.3	Gestión de progreso	jun 11	jun 22	10d
1.1.4	Gestión de interfaz	jun 25	jul 6	10d
1.1.5	Etapa de pruebas intermedia	jul 9	jul 9	1d
1.1.6	Documentación de la implementación y pruebas	jul 10	jul 10	1d
1.2	**Iteración C2**	**jul 11**	**ago 21**	**19d**
1.2.1	Planificación de la fase de construcción - C2	jul 11	jul 11	1d
1.2.2	Revisión de los módulos implementados en C1	jul 12	jul 12	1d
1.2.3	Gestión de problemas	jul 13	jul 19	5d
1.2.4	Gestión de navegación	jul 20	ago 9	5d
1.2.5	Gestión de usuarios	ago 10	ago 17	5d
1.2.6	Etapa de pruebas	ago 20	ago 20	1d
1.2.7	Documentación de la implementación y pruebas	ago 21	ago 21	1d
1.3	**Iteración C3**	**ago 22**	**ago 27**	**4d**
1.3.1	Planificación de la fase de construcción - C3	ago 22	ago 22	1d
1.3.2	Revisión de los módulos implementados en C2	ago 23	ago 23	1d
1.3.3	Etapa de pruebas	ago 24	ago 24	1d
1.3.4	Documentación de las pruebas	ago 27	ago 27	1d
1.4	**Iteración C4**	**ago 28**	**sep 17**	**14d**
1.4.1	Planificación de la fase de construcción - C4	ago 28	ago 28	1d
1.4.2	Gestión de cursos	ago 29	sep 5	6d
1.4.3	Etapa de pruebas	sep 6	sep 10	3d
1.4.4	Definición del prototipo	sep 12	sep 13	2d
1.4.5	Documentación de la implementación y pruebas	sep 14	sep 14	1d
1.4.6	Preparación de la fase de transición	sep 17	sep 17	1d
1.4.7	Entrega del informe previo	sep 17	sep 17	N/D

Figura B.9: Fechas de la fase de construcción.

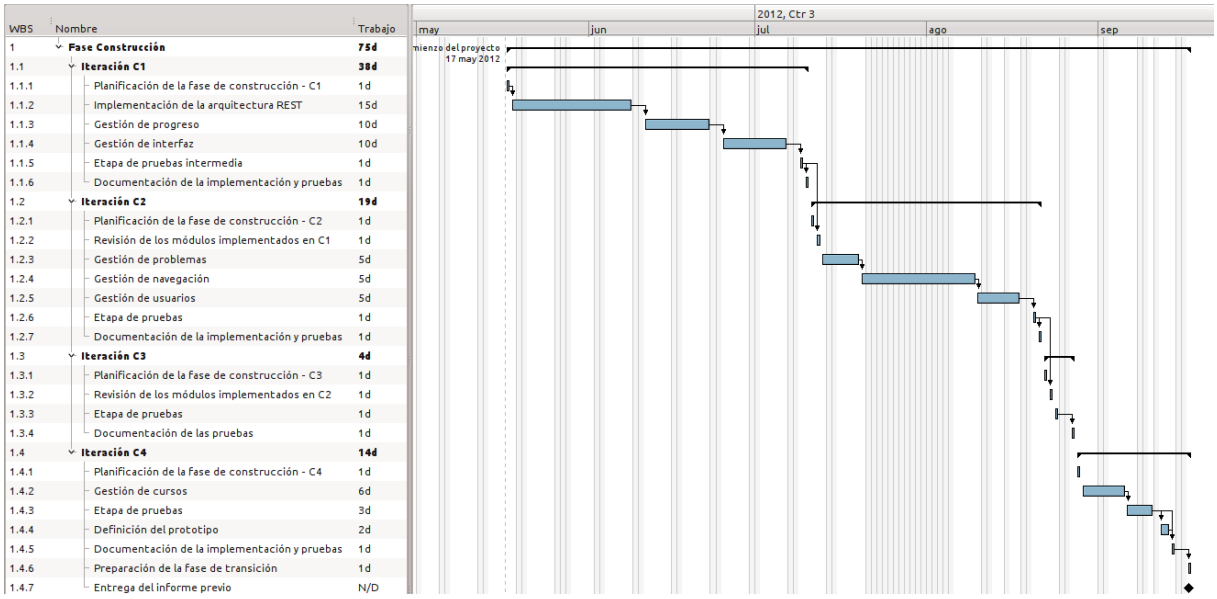
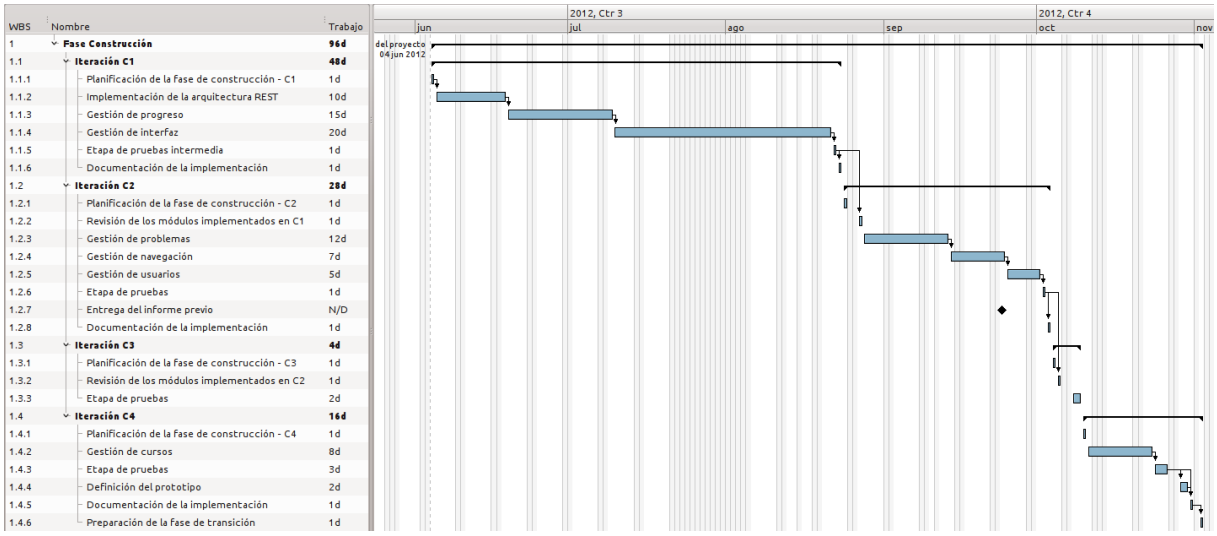


Figura B.10: Distribución temporal de la fase de construcción.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Construcción**	**jun 4**	**nov 2**	**96d**
1.1	**Iteración C1**	**jun 4**	**ago 23**	**48d**
1.1.1	Planificación de la fase de construcción - C1	jun 4	jun 4	1d
1.1.2	Implementación de la arquitectura REST	jun 5	jun 18	10d
1.1.3	Gestión de progreso	jun 19	jul 9	15d
1.1.4	Gestión de interfaz	jul 10	ago 21	20d
1.1.5	Etapa de pruebas intermedia	ago 22	ago 22	1d
1.1.6	Documentación de la implementación	ago 23	ago 23	1d
1.2	**Iteración C2**	**ago 24**	**oct 3**	**28d**
1.2.1	Planificación de la fase de construcción - C2	ago 24	ago 24	1d
1.2.2	Revisión de los módulos implementados en C1	ago 27	ago 27	1d
1.2.3	Gestión de problemas	ago 28	sep 13	12d
1.2.4	Gestión de navegación	sep 14	sep 24	7d
1.2.5	Gestión de usuarios	sep 25	oct 1	5d
1.2.6	Etapa de pruebas	oct 2	oct 2	1d
1.2.7	Entrega del informe previo	sep 24	sep 24	N/D
1.2.8	Documentación de la implementación	oct 3	oct 3	1d
1.3	**Iteración C3**	**oct 4**	**oct 9**	**4d**
1.3.1	Planificación de la fase de construcción - C3	oct 4	oct 4	1d
1.3.2	Revisión de los módulos implementados en C2	oct 5	oct 5	1d
1.3.3	Etapa de pruebas	oct 8	oct 9	2d
1.4	**Iteración C4**	**oct 10**	**nov 2**	**16d**
1.4.1	Planificación de la fase de construcción - C4	oct 10	oct 10	1d
1.4.2	Gestión de cursos	oct 11	oct 23	8d
1.4.3	Etapa de pruebas	oct 24	oct 26	3d
1.4.4	Definición del prototipo	oct 29	oct 30	2d
1.4.5	Documentación de la implementación	oct 31	oct 31	1d
1.4.6	Preparación de la fase de transición	nov 2	nov 2	1d

Figura B.11: Fechas reales de la fase de construcción.



WBS	Nombre	Trabajo
1	**Fase Construcción**	**96d**
1.1	**Iteración C1**	**48d**
1.1.1	Planificación de la fase de construcción - C1	1d
1.1.2	Implementación de la arquitectura REST	10d
1.1.3	Gestión de progreso	15d
1.1.4	Gestión de interfaz	20d
1.1.5	Etapa de pruebas intermedia	1d
1.1.6	Documentación de la implementación	1d
1.2	**Iteración C2**	**28d**
1.2.1	Planificación de la fase de construcción - C2	1d
1.2.2	Revisión de los módulos implementados en C1	1d
1.2.3	Gestión de problemas	12d
1.2.4	Gestión de navegación	7d
1.2.5	Gestión de usuarios	5d
1.2.6	Etapa de pruebas	1d
1.2.7	Entrega del informe previo	N/D
1.2.8	Documentación de la implementación	1d
1.3	**Iteración C3**	**4d**
1.3.1	Planificación de la fase de construcción - C3	1d
1.3.2	Revisión de los módulos implementados en C2	1d
1.3.3	Etapa de pruebas	2d
1.4	**Iteración C4**	**16d**
1.4.1	Planificación de la fase de construcción - C4	1d
1.4.2	Gestión de cursos	8d
1.4.3	Etapa de pruebas	3d
1.4.4	Definición del prototipo	2d
1.4.5	Documentación de la implementación	1d
1.4.6	Preparación de la fase de transición	1d

Figura B.12: Distribución temporal real de la fase de construcción.

B.4. Fase de transición

El objetivo principal de la fase de transición es la transición del sistema implementado a la instalación del sistema en el entorno definitivo de producción, así como garantizar que los usuarios del producto pueden usarlo sin problemas. Al final de esta fase, el producto debe estar funcionando.

B.4.1. Planificación temporal mediante un diagrama de *Gantt*

B.4.1.1. Planificación previa

Debido a que la fase consta de una única iteración, se establece como fecha de inicio el martes 18 de septiembre de 2012 y como fecha final el jueves 22 de noviembre de 2012, tal y como muestran las imágenes B.13 y B.14.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Transición**	**sep 18**	**nov 22**	**45d**
1.1	**Iteración T1**	**sep 18**	**nov 22**	**45d**
1.1.1	Planificación de la fase de transición	sep 18	sep 18	1d
1.1.2	Etapa de pruebas externas	sep 19	sep 25	5d
1.1.3	Detección de cambios	sep 26	sep 27	2d
1.1.4	Aplicar cambios	sep 28	oct 4	5d
1.1.5	Etapa final de pruebas	oct 5	oct 16	7d
1.1.6	Definición de la versión final del producto	oct 17	oct 17	1d
1.1.7	Documentación del sistema	oct 18	oct 24	5d
1.1.8	Documentación de ayuda al usuario	oct 25	oct 26	2d
1.1.9	Realización de la memoria del proyecto	oct 29	nov 19	15d
1.1.10	Revisar la coherencia "proyecto-documentación"	nov 20	nov 21	2d
1.1.11	Entrega de la versión final del producto	nov 22	nov 22	N/D
1.1.12	Entrega de la versión final de la memoria	nov 22	nov 22	N/D

Figura B.13: Fechas de la fase de transición.

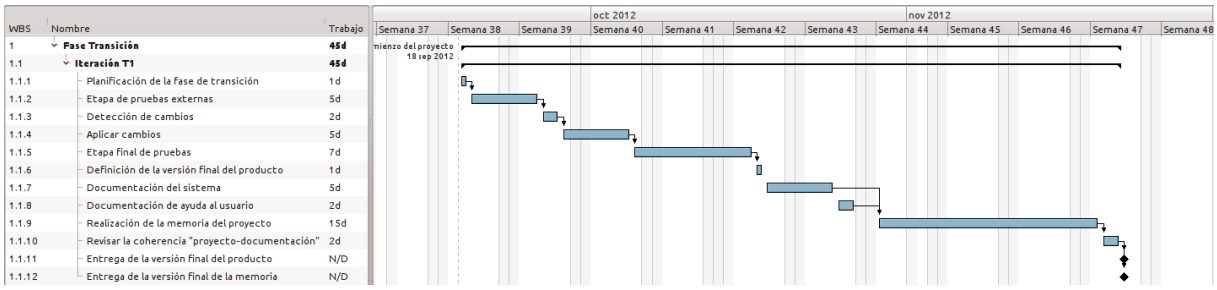


Figura B.14: Distribución temporal de la fase de transición.

B.4.1.2. La realidad de la fase de transición

Esta fase se ha realizado en cinco días menos de lo previsto esencialmente porque las pruebas han arrojado pocos cambios a arreglar en el sistema, haciendo que las etapas de pruebas y las de aplicación de estos cambios fueran más cortas de lo inicialmente planificado.

WBS	Nombre	Inicio	Fin	Duración
1	**Fase Transición**	**nov 5**	**ene 4**	**40d**
1.1	**Iteración T1**	**nov 5**	**ene 4**	**40d**
1.1.1	Planificación de la fase de transición	nov 5	nov 5	1d
1.1.2	Etapa de pruebas externas	nov 6	nov 7	2d
1.1.3	Detección de cambios	nov 8	nov 8	1d
1.1.4	Aplicar cambios	nov 9	nov 15	5d
1.1.5	Etapa final de pruebas	nov 16	nov 22	5d
1.1.6	Definición de la versión final del producto	nov 23	nov 23	1d
1.1.7	Documentación del sistema	nov 26	nov 29	4d
1.1.8	Documentación de ayuda al usuario	nov 30	nov 30	1d
1.1.9	Realización de la memoria del proyecto	dic 3	dic 31	18d
1.1.10	Revisar la coherencia "proyecto-documentación"	ene 2	ene 3	2d
1.1.11	Entrega de la versión final del producto	ene 4	ene 4	N/D
1.1.12	Entrega de la versión final de la memoria	ene 4	ene 4	N/D

Figura B.15: Fechas reales de la fase de transición.

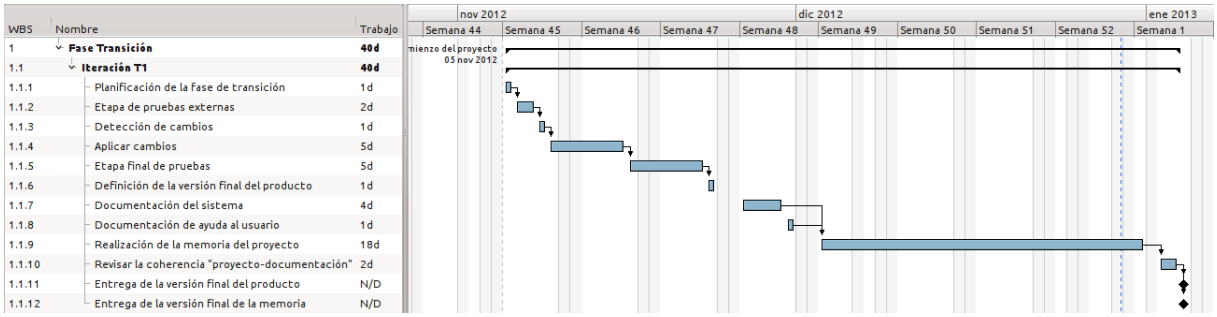


Figura B.16: Distribución temporal real de la fase de transición.

B.5. Ajuste a la planificación

La valoración final del ajuste a la planificación es bueno, ya que el retraso acumulado del proyecto ha estado de 26 días de trabajo (43 días naturales, contando los fines de semana y festivos).

Existen dos motivos principales que justifican la fluctuación en los plazos de entrega. El retraso total del tiempo de entrega se ha contenido por haber eliminado (o cambiado) ciertas funcionalidades del sistema, tal y como se detalla en el apartado 7.2. Pero no hay que olvidar que la realización de la memoria del proyecto y la documentación para usuarios y para desarrolladores ha supuesto un mayor esfuerzo del previsto.

En resumen, se trata de un retraso aceptable con leves repercusiones en la planificación inicial, que en ningún caso se verán repercutidas en el coste económico del proyecto (ver apartado 4.4.2.3 para más información).

Apéndice C

Glosario

El propósito de este anexo es recopilar, analizar y definir el vocabulario necesario para una correcta comprensión global del proyecto a desarrollar. Su redacción ha sido pensada tanto para los miembros encargados del desarrollo como para los usuarios finales.

Aspecto Elemento, faceta o matiz de algo. En este documento se refiere a los distintos matices de cada ítem a estudiar (ver apartados 6.7.2 y 6.7.3).

Catálogo de cursos El conjunto de cursos que se disponen en la aplicación para ser estudiados a través de programas de estudio y de ejercicios. En este proyecto, principalmente, vocabulario en inglés.

Cloud Computing Paradigma computacional que se basa en el consumo de recursos, ya sean aplicaciones, computación o *hardware*, ofrecido mediante una red, usualmente Internet. Este consumo de recursos se realiza bajo demanda, como si de un servicio se tratara (agua, gas...). Estos servicios pueden ser públicos o privados, gratuitos o de pago, e incorporan acuerdos de tipo `SLA`. Las distintas capas que componen este paradigma son: *software*, plataforma e infraestructura como servicio.

Favicon En castellano, icono de página (del inglés *favorites icon*) es una pequeña imagen asociada con una página *web* en particular.

***Frequently Asked Questions* (`FAQ`)** Lista de preguntas y respuestas, las cuales son comúnmente preguntadas en un contexto determinado y pertenecientes a un tema en particular.

Herramientas de trabajo En este proyecto se utilizan herramientas de comunicación y compartición (el correo electrónico, *DropBox*, *Google Docs*, *Google App Engine*, LaTeX, *Mercurial*, *BitBucket*...), principalmente. En el anexo D hay información más detallada de cada una de ellas.

***Infraestructure as a Service* (`IaaS`)** En castellano, infraestructura como servicio. Medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. Servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran para manejar tipos específicos de cargas de trabajo de forma externa a la empresa que contrata esta infraestructura. Un ejemplo de este servicio es *Amazon S3*.

Ítem Cada uno de los elementos que forman parte de un dato. En este documento se refiere a cada concepto que el usuario deberá aprender, del cual deberá conocer todos y cada uno de los aspectos que lo definen. En la fase de construcción cambió el concepto; ver definición de Problema.

Material Todo el conjunto de información disponible en la aplicación para ser estudiado. Se distribuye entre catálogo de cursos, series de estudio y problemas.

***Platform as a Service* (PaaS)** En castellano, plataforma como servicio. Conjunto de soluciones, o también grupo de sistemas y componentes necesarios para el desarrollo de un producto funcional y robusto, ofrecidos como servicio dentro del paradigma de *cloud computing*. Este servicio es realmente útil para desarrolladores ya que facilita el desarrollo de aplicaciones sin la necesidad de adquirir *hardware* dedicado para ello y con capacidades de aprovisionamiento y de alojamiento. Además se ofrecen facilidades para la realización de pruebas, diseño, aplicaciones colaborativas, integración de bases de datos, seguridad, etc. Ejemplos de este tipo de servicio son *Google App Engine* o *Amazon EC2*, entre otros.

Problema Cada uno de los conceptos que el usuario ha de aprender incluido en un curso.

Progreso Avance, adelante, perfeccionamiento. En este proyecto se refiere al perfeccionamiento que tiene cada usuario respecto los cursos que está estudiando en cada una de las sesiones de estudio.

Proveedor de servicios de Internet Empresa dedicada a proporcionar conexión a Internet a los usuarios que contraten dicho servicio, garantizando el correcto funcionamiento del mismo.

Requisito Circunstancia o condición necesaria para algo.

Requisito funcional Define el comportamiento interno del *software*: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica. La implementación de los requisitos funcionales se detalla en el apartado 2.8.1. En resumen, se podría decir que se incluyen dentro del "qué ha de hacer" un sistema.

Requisito no funcional Requisitos que se centran en el diseño o la implementación del sistema. La implementación de los requisitos no funcionales se detalla en el apartado 2.8.2. En resumen, se podría decir que los requisitos no funcionales se incluyen dentro del "cómo ha de ser" un sistema.

Rol Papel. Función que algo o alguien cumple. En este proyecto, se refiere a las distintas funciones que jugarán los usuarios del sistema, tanto técnicos como usuarios.

***Rational Unified Process* (RUP)** Proceso de desarrollo de programas. Constituye una de las metodologías de desarrollo más utilizadas, en colaboración con UML.

Script Programa generalmente simple que, por lo general, se almacena en un archivo de sólo texto.

Sesión de estudio Conjunto de problemas que comparten características comunes. Un ejemplo de sesión de estudio sería, dentro de la materia de inglés, el conjunto de

ejercicios que se centran en la fonética de las palabras para comprobar si el usuario es capaz de distinguir las distintas pronunciaciones y ubicar cada palabra en su contexto adecuado.

***Service Level Agreement* (SLA)** En castellano, acuerdo de nivel de servicio. Contrato escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio, entendiendo como tal: tiempo de respuesta, disponibilidad horaria, documentación disponible, personal asignado al servicio, etc.

Sistema de información Conjunto de elementos orientados al tratamiento y administración de datos e información, organizados y listos para su posterior uso, generados para cubrir una necesidad (objetivo). Dichos elementos formarán parte de alguna de estas categorías: personas, datos, actividades o técnicas de trabajo y recursos materiales en general.

***Software as a Service* (SaaS)** En castellano, *software* como servicio. Modelo de distribución de *software* donde éste y los datos que necesita se alojan en servidores ajenos al usuario final de la aplicación, al cual se accede mediante un navegador *web* o un *thin client*. A este *software* se puede acceder desde cualquier equipo, sin necesidad de realizar ninguna instalación adicional para su funcionamiento. Un ejemplo de este tipo de servicio es *Google Docs*.

Stakeholders Persona o entidad que están o pueden estar afectadas por las actividades del proyecto. Además, es el público interesado, y son esenciales en la planificación estratégica del negocio.

***Unified Modeling Language* (UML)** En castellano, lenguaje de modelado de sistemas de *software*. Es el lenguaje de modelado de sistemas *software* más conocido y utilizado en la actualidad.

***Uniform Resource Identifier* (URI)** En castellano, identificador uniforme de recurso. Es una cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

***Uniform Resource Locator* (URL)** En castellano, localizador uniforme de recurso. En esencia, no es más que una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización como, por ejemplo documentos de texto, imágenes, vídeos, presentaciones, etc. El URL de un recurso de información es su dirección en *Internet*, la cual permite que el navegador la encuentre y la muestre de forma adecuada.

Universidad Politécnica de Catalunya Institución pública de investigación y educación superior, especializada en los ámbitos de arquitectura, ciencias e ingeniería. También se la conoce por sus siglas UPC y, más recientemente, por la marca "Barcelona Tech".

Usuario Principal beneficiario del sistema. Cualquier persona que haga uso de la aplicación para el fin que ha sido diseñada.

Apéndice D

Herramientas utilizadas

Durante el desarrollo de este proyecto se han utilizado las siguientes herramientas:

BitBucket Servicio de alojamiento basado en *web* para los proyectos que utilizan el sistema de control de versiones *Mercurial*. Ofrece cuentas gratuitas con un número ilimitado de repositorios privados. El servicio está escrito en *Python*. El servicio está disponible en `https://bitbucket.org/`.

Dropbox Herramienta multiplataforma que facilita compartir y almacenar ficheros en línea. En este proyecto, su principal uso es el de almacenar la documentación generada en cada iteración. Se puede acceder a través del enlace `https://www.dropbox.com`.

Google App Engine Plataforma como servicio de *cloud computing* para desarrollo y alojamiento de aplicaciones *web* gestionadas en *Google*. Se puede visitar `http://appengine.google.com/` para usar el servicio.

Google Docs Programa gratuito basado en *web* para crear documentos en línea con la posibilidad de colaborar en grupo. Incluye un procesador de textos, una hoja de cálculo, un programa de presentación básico y un editor de formularios destinados a encuestas. En este proyecto se usa para la edición y generación de documentación, previo paso a su almacenamiento en *DropBox*. Su uso es posible, previo registro, a través de `https://docs.google.com`.

Kile Se ha utilizado este entorno de edición de LaTeX integrado en el gestor `KDE`. Este programa y todos los paquetes utilizados son libres.

Mercurial Sistema de control de versiones multiplataforma para desarrolladores de *software*. Está implementado principalmente en *Python*. Todas las operaciones de *Mercurial* se invocan mediante línea de comandos y como opciones dadas a su programa motor, `hg`, que hace referencia al símbolo químico del mercurio.

MockFlow El servicio accesible a través de la dirección `http://www.mockflow.com/` permite diseñar bocetos de interfaces gráficas para programas y páginas web.

Mozilla Firefox El navegador que permite acceder a casi todas las herramientas anteriormente descritas. Se puede descargar de `http://www.firefox.com`. En esta ocasión, se ha utilizado la extensión *Web Developer* para poder desarrollar el sistema de una manera correcta y multiplataforma.

Planner Un gestor de proyectos que se integra en el gestor `GNOME`. Se puede encontrar el programa y su documentación en la página `http://live.gnome.org/Planner`.

Phyton Lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible. Es el lenguaje en el que se ha desarrollado la aplicación. Se puede acceder a través del enlace `http://www.python.org/`.

yEd Editor de diagramas, basado en *Java*, que se puede ejecutar en cualquier sistema operativo. Lo ha creado "yWorks GmbH" y ellos poseen sus derechos de autor, aunque es totalmente gratuito. Se puede descargar desde la página de los autores, que es `http://www.yworks.com/en/products_yed_about.html`.

Apéndice E

Modelo de datos con funciones

En este anexo se presenta el modelo de datos normalizado que añade las funciones implementadas en cada clase.

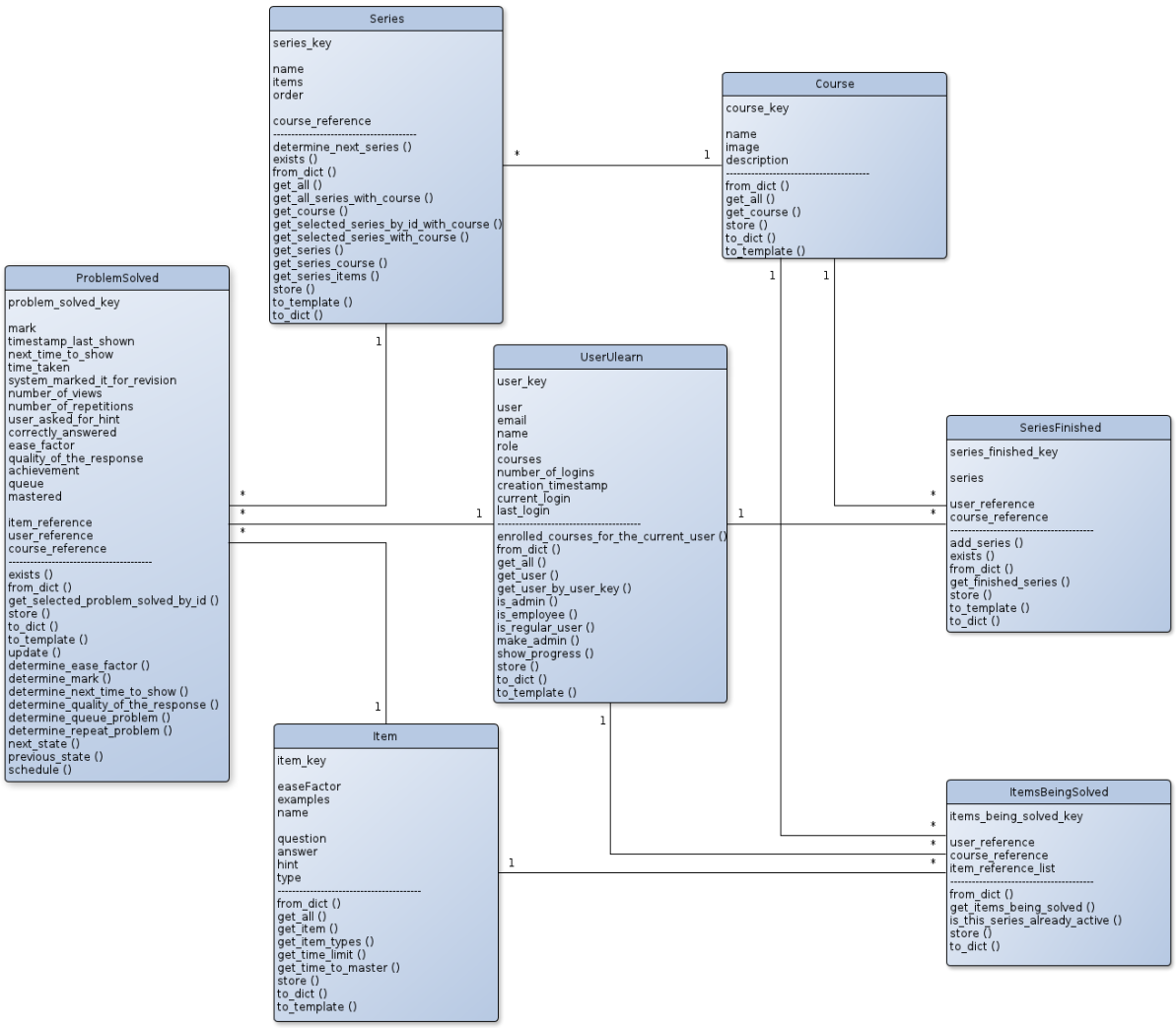


Figura E.1: Modelo de datos detallando las funciones de cada clase.

Apéndice F

Logotipos oficiales

Aquí se muestran los logotipos de la aplicación **ULearn** cuando se accede a la página principal o a una página no encontrada, respectivamente (tal y como se detalla en el apartado 2.8.2.6). También se muestra el `favicon` que se diseñó para los navegadores.

Figura F.1: Logotipo de ULearn.

Figura F.2: Logotipo que aparece cuando se accede a una página no encontrada.

Figura F.3: `Favicon` de ULearn.

Bibliografía

[1] Paul Barry. *Head First Python*. O'Reilly Media, November 2010.

[2] Dolors Costal, Xavier Franch, Maria Ribera Sancho, and Ernest Teniente. *Enginyeria del software: Especificació*. Edicions UPC, Barcelona, 2005.

[3] Roger L. Costello. Building Web Services the REST Way. Página web. http://www.xfront.com/REST-Web-Services.html.

[4] Google App Engine Developers. ¿Por qué App Engine? Página web, 2012. http://code.google.com/intl/es-ES/appengine/whyappengine.html.

[5] The Rational Edge. The Ten Essentials of RUP. Página web, 2003. http://www.devarticles.com/c/a/Development-Cycles/The-Ten-Essentials-of-RUP/.

[6] Dr. M. Elkstein. Learn REST: A Tutorial. Página web, 2008. http://rest.elkstein.org/.

[7] Charles Fishman. *They Write the Right Stuff*. Johnson Space Center Shuttle Software Group, Issue 6, p. 95, December 1996.

[8] Joe Gregorio. How to Create a REST Protocol. Página web, 2004. http://www.xml.com/pub/a/2004/12/01/restful-web.html.

[9] Steve Klabnik. Haters gonna HATEOAS. Página web. http://timelessrepo.com/haters-gonna-hateoas.

[10] Mike Pearce. Cookies and the RESTful API. Página web, 2010. http://blog.mikepearce.net/2010/08/24/cookies-and-the-restful-api/.

[11] Charles Severance. *Using Google App Engine*. O'Reilly Media and Google Press, May 2009.

[12] Kevan Stannard. How to use sessions on Google App Engine with Python and gae-sessions? Página web, 2011. http://blog.stannard.net.au/2011/01/09/how-to-use-sessions-on-google-app-engine-with-python-and-gae-sessions/.

[13] Stefan Tilkov. A Brief Introduction to REST. Página web, 2010. http://www.infoq.com/articles/rest-introduction.

[14] Malmö University. Artifacts. Página web, 2001. http://www.ts.mah.se/RUP/RationalUnifiedProcess/process/artifact/ovu_arts.htm.

[15] Malmö University. Introducción a RUP. Página web, 2001. http://www.ts.mah.se/RUP/RationalUnifiedProcess/.

[16] Jim Webber, Savas Parastatidis, and Ian Robinson. *REST in Practice*. O'Reilly Media, September 2010.

[17] SuperMemo World. History of SuperMemo. Página web, 2009. http://www.supermemo.com/english/history.htm.